THE UNIVERSITY of EDINBURGH

Advances in Interior Point Methods and Column Generation

Pablo González Brevis

Doctor of Philosophy
University of Edinburgh
2013

Declaration

I declare that this thesis was composed by myself and that the work contained therein is my own, except where explicitly stated otherwise in the text.

(*Pablo González Brevis*)

To my endless inspiration and angels on earth: Paulina, Cristóbal and Esteban

Abstract

In this thesis we study how to efficiently combine the column generation technique (CG) and interior point methods (IPMs) for solving the relaxation of a selection of integer programming problems. In order to obtain an efficient method a change in the column generation technique and a new reoptimization strategy for a primal-dual interior point method are proposed.

It is well-known that the standard column generation technique suffers from unstable behaviour due to the use of optimal dual solutions that are extreme points of the restricted master problem (RMP). This unstable behaviour slows down column generation so variations of the standard technique which rely on interior points of the dual feasible set of the RMP have been proposed in the literature. Among these techniques, there is the primal-dual column generation method (PDCGM) which relies on sub-optimal and well-centred dual solutions. This technique dynamically adjusts the column generation tolerance as the method approaches optimality. Also, it relies on the notion of the symmetric neighbourhood of the central path so sub-optimal and well-centred solutions are obtained. We provide a thorough theoretical analysis that guarantees the convergence of the primal-dual approach even though sub-optimal solutions are used in the course of the algorithm. Additionally, we present a comprehensive computational study of the solution of linear relaxed formulations obtained after applying the Dantzig-Wolfe decomposition principle to the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST). We compare the performance of the PDCGM with the standard column generation method (SCGM) and the analytic centre cutting planning method (ACCPM). Overall, the PDCGM achieves the best performance when compared to the SCGM and the ACCPM when solving challenging instances from a column generation perspective. One important characteristic of this column generation strategy is that no specific tuning is necessary and the algorithm poses the same level of difficulty as standard column generation method. The natural stabilization available in the PDCGM due to the use of sub-optimal well-centred interior point solutions is a very attractive feature of this method. Moreover, the larger the instance, the better is the relative performance of the PDCGM in terms of column generation iterations and CPU time.

The second part of this thesis is concerned with the development of a new warmstarting strategy for the PDCGM. It is well known that taking advantage of the previously solved RMP could lead to important savings in solving the modified RMP. However, this is still an open question for applications arising in an integer optimization context and the PDCGM. Despite the current warmstarting strategy in the PDCGM working well in practice, it does not guarantee full feasibility restorations nor considers the quality of the warmstarted iterate after new columns are added. The main motivation of the design of the new warmstarting strategy presented in this thesis is to close this theoretical gap. Under suitable assumptions, the warmstarting procedure proposed in

this thesis restores primal and dual feasibilities after the addition of new columns in one step. The direction is determined so that the modification of small components at a particular solution is not large. Additionally, the strategy enables control over the new duality gap by considering an expanded symmetric neighbourhood of the central path. As observed from our computational experiments solving CSP and VRPTW, one can conclude that the warmstarting strategies for the PDCGM are useful when dense columns are added to the RMP (CSP), since they consistently reduce the CPU time and also the number of iterations required to solve the RMPs on average. On the other hand, when sparse columns are added (VRPTW), the coldstart used by the interior point solver HOPDM becomes very efficient so warmstarting does not make the task of solving the RMPs any easier.

Acknowledgements

I would like to thank Jacek Gondzio for being such a wonderful supervisor. I have learnt a lot from him and for that I am eternally grateful.

I would like to express my gratitude to my examiners, Julian Hall and John Mitchell for their insightful comments and suggestions about this work.

I would like to thank Universidad del Desarrollo, particularly the Research Council and the School of Engineering, for their financial support. Also, I am grateful to MIDEPLAN and CONICYT for granting me the *Presidente de la República Scholarship*. I would like to thank the School of Mathematics at The University of Edinburgh for its financial support during the last six months of my studies.

I am also very grateful to everyone who supported me and my family during this "adventure". Specially, I am indebted to Nora Au and Pedro Silva for their constant support from Chile since the beginning.

I would also like to thank all my office mates, Bubacarr, Naiyuan, Kimonas, Robert, Tim, Hanyi and Luciana. It was great sharing room 5620 with you guys. Special thanks go to Pedro, Robert and Tim for proofreading a first draft of this thesis.

I am also indebted to the staff and administrators at the School of Mathematics at The University of Edinburgh. Particularly, friends and colleges at the Edinburgh Research Group on Optimization (ERGO).

I would like to thank all baristas at the Starbucks store at Forrest Road. You always made my mornings (and some afternoons) with a delicious mocha.

In a more personal note, I am very grateful to friends I had the chance to meet while in Edinburgh. To Cristian, Loreto, Bryan, Lorena, Pedro, Héctor, Valeria and Milan, you are wonderful people!

I would like to thank my parents, Verónica and Enrique; my brothers and sister, Claudio, Sebastián and Patricia; and my parents-in-law, Carolina and Julio, for their constant support from far away. You have always believed in me!

Last but not least, this thesis could not have been completed without the support of my family. To Cristóbal and Esteban for showing me the beauty on simple things. To Paulina, for her constant encouragement, endless patience and infinite love!

A todos, gracias totales!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter we give a brief introduction to linear programming and some solution techniques to solve this class of problems. We also state the scope and the main contributions of this thesis. At the end of the chapter, we present the outline of this document.

1.1 Linear programming problems

Linear programming or linear optimization is regarded as one of the most well-established fields in optimization due to its long history, understanding and wide range of solvers and methodologies available to tackle it. The linear programming problem is a special case of the general mathematical optimization problem in which an objective function f is minimized over a feasible set $G \subseteq \mathbb{R}^n$ which is described as

$$\min\{f(x) : x \in G\}, \tag{1.1}$$

where $f : G \to \mathbb{R}$ [93, 101]. In its linear case, $f(x)$ is a linear function and G is a convex feasible set defined by linear (in)equalities. Linear programming problems can be found in a wide spectrum of real-life applications such as transportation, economics, energy and logistics among several others. More sophisticated applications are in the context of non-linear programming problems where in many cases the non-linearities are approximated by linear functions and therefore one ends up solving a sequence of linear programming problems. Also, very interesting applications are integer and combinatorial optimization problems in which linear continuous relaxations are commonly used to solve these problems [100].

Among the preferred methods to solve linear programming problems are the *simplex method* and *interior point methods*.

Simplex method. This method was introduced by Dantzig [23] and aims to find an optimal solution by using partitions of the variables into basic and nonbasic sets, where the variables in the former set may take non-negative values while the variables included in the latter set are set to zero. The simplex method starts from a vertex of the polytope describing the feasible set G and seeks a direction that improves the value of the objective function by checking the rate of change of the objective in the directions of the neighbouring vertices. Since only neighbouring vertices are considered, directions along the edges of this polytope are obtained. In this sense, the simplex method has a very limited vision of the complete polytope and may prove to be inefficient for some

problems, as shown by Klee and Minty in [81]. Although the theoretical worst case complexity bound on the possible number iterations is exponential in the dimension of the vector x, contemporary codes are stunningly efficient and simplex is regarded as one of the major breakthrough in optimization [67, 68] and is the default choice of commercial/academic solvers such us CPLEX [71] and XPRESS [38].

Interior point methods A completely different approach was proposed by Dikin [30] and then developed by Karmarkar [77]. In contrast to the simplex method in which every iterate is an extreme point of the feasible polytope G, these algorithms generate a sequence of iterates which traverse the relative interior of the feasible region. Several variants of interior point methods (IPMs) offer polynomial complexity such as the *projective methods*, *path-following methods* and *affine potential reduction methods* [66]. Among these variants, a family of path-following primal-dual methods has proven to be the most important and widely used [121]. Additionally, interior point methods also known as *barrier methods* due to the use of *logarithmic barriers* replacing non-negative constraints, have been shown to be frequently more efficient in general than simplex methods for solving large scale problems [87, 92].

Another powerful tool used in solving large scale linear programming problems, known as *column generation*, has become an important technique in past decades [28, 118]. Although this technique is not a solver itself as simplex or interior point methods are, column generation is an iterative algorithm that is commonly used in the context of integer programming to solve problems with a large number of variables. This technique was introduced by Dantzig and Wolfe in [26] when solving a linear programming problem with a large number of variables in a column-wise manner. This was shortly after Gilmore and Gomory [46, 47] proposed this technique to solve the cutting stock problem. This technique is an iterative procedure applied to solve a linear programming problem with a possibly 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 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 few columns of the MP are considered at first. Iteratively, new columns with negative reduced costs are generated and added to the RMP. These columns are given by one or more *pricing subproblems*. In general the method converges to an optimal solution by generating a relatively small subset of columns. Note that at termination the RMP is equivalent to the original problem (MP), and therefore, the optimal solution of the RMP is the optimal solution of the MP.

From a dual point of view, the dual of the RMP is a relaxation of the dual MP (*i.e.,* problem with fewer constraints). Hence, the process which generates the columns in the primal space (pricing subproblems) can be also interpreted as the process which generates cuts in the dual space (separation routine). This cut generation method applied to the dual is known as Kelley's *cutting plane method* [78]. Due to this primal-dual correspondence, column generation and cutting plane methods are equivalent.

1.2 Motivation and contributions

Although interior point methods, column generation and cutting plane methods have proven to be powerful techniques for solving large linear programming problems, few

attempts have been made to combine them efficiently. Among these attempts one could mention [34, 49, 62, 89, 93, 96].

Despite the benefits showed by these methods for particular applications, somehow these efforts still have not been widely taken up by the column generation community and simplex-based methods are still the preferred techniques in this context.

In [86], one of the key papers in column generation, the following statement can be found:

> *Besides subgradients and simplex-based methods, the Lagrangian dual can be solved with more advanced (nonlinear) alternatives with stronger convergence properties. Among them are the bundle method [69] based on quadratic programming, and the analytic center cutting-plane method [53], an interior point solution approach. However, the performance of these alternatives is still to be evaluated in the context of integer programming.*

Moreover, in the same article Lübbecke and Desrosiers state:

> *For some linear programs barrier methods [15] can prove most effective, although there is no possible warm start.*

Although several techniques have been proposed in order to reoptimize IPMs in the context of cutting plane/column generation [51, 52, 56] these developments have not yet managed to change the views of the (conservative) column generation community.

We believe that some of the reasons why people have been discouraged from using interior point methods in column generation are the following:

- **Historical considerations.** Interior point methods are novel techniques when compared to the traditional and well-established simplex method. Additionally, when column generation and cutting plane methods were proposed in the early 60's, simplex was the only efficient method at hand while IPMs have matured only in recent years.

- **Reoptimization**. In the column generation/cutting plane framework, solving relatively similar problems can be exploited, saving iterations and time and therefore making the whole iterative process more efficient. Reoptimizing with simplex is done very efficiently and very intuitively by keeping primal/dual feasibility and seeking feasibility in the dual/primal space. Reoptimizing with interior point methods require more "sophisticated" techniques and a good understanding of this and efficient implementations have only been proposed in recent years.

- **"Simplexification" of IPMs.** Replacing the simplex method by IPMs in the column generation/cutting plane context has shown mixed results and tends to favour the simplex method since column generation was developed with this method in mind. On top of that, attempting a naive reoptimization with IPMs using the optimal solution on the boundary of the polytope G, such as it is done with the simplex method has proven to be disastrous. Therefore, the incentive to move from a well-established method such as simplex to a method that allows some doubts is null.

Having all these considerations in mind, the main goal of this thesis is to show how to combine column generation and interior point methods efficiently in the context of integer programming for solving the relaxation of a selection of combinatorial optimization problems commonly used by the column generation community.

This requires a change of the column generation paradigm and the way reoptimization is performed with IPMs.

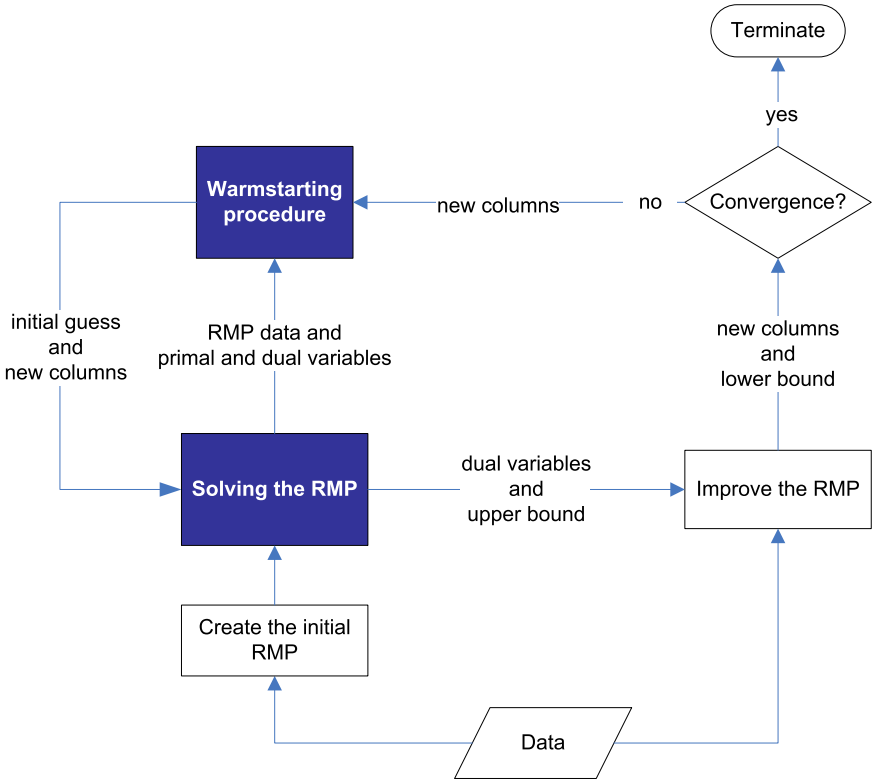


Figure 1.1: Solving the MP - Column generation flow chart considering a warmstarting procedure.

In order to facilitate the understanding of the main contributions of this thesis in the context of column generation, in Figure 1.1 we have a schematic diagram showing the principal components of this technique. In this thesis we are interested in the coloured components of Figure 1.1: (i) how to solve the RMPs so better dual variables are sent to the procedure that generates new columns (denoted by "Improve the RMP"); and (ii) how to provide an initial guess such as the RMP is solved more efficiently than when no prior information is used. Column generation is fully described in Chapter 3.

In [34], Elhedhli and Goffin study how to compute a Lagrangean-based lower bound using the analytic centre cutting plane method (ACCPM) [49]. Also, they study how to use the information of a parent node (cuts and lower bounds) in order to speed up the solution at a child node inside a Lagrangean-based branch-and-bound scheme. In a similar spirit but without the branching stage, our study considers the main components of a column generation scheme that uses the primal-dual column generation method proposed in [62] to obtain solutions of the RMP, instead of the ACCPM.

Below, we describe in more detail the motivations and major outcomes the reader will find in this thesis.

It has been observed that in the standard column generation technique, unstable behaviour is caused by the use of optimal dual solutions that are extreme points of the RMPs [86, 117]. This unstable behaviour slows down the column generation method so variations of the standard technique relying on interior points of the dual feasible set of the RMP have been proposed in the literature [17, 86].

A prime focus of this thesis is on the primal-dual column generation technique (PDCGM) proposed by Gondzio and Sarkissian [62], where an interior point method is used to obtain sub-optimal solutions that are well-centred in the dual feasible set of the corresponding RMP. Promising computational results have been reported using this technique [62, 89], but for a very limited class of problems. Moreover, the PDCGM 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 this primal-dual approach has been presented. One of the objectives of this thesis is to provide these guarantees. In order to achieve this goal, we first review the primal-dual column generation technique and show that it converges to an optimal solution of the master problem if such exists, even though sub-optimal solutions are used in the course of the algorithm.

Additionally, to demonstrate how the method compares with other column generation approaches, we present a comprehensive computational performance study for solving linear-relaxed formulations obtained after applying the Dantzig-Wolfe decomposition principle [26] for three classes of problems which are well-known in the column generation literature: the cutting stock problem (CSP) [46], the vehicle routing problem with time windows (VRPTW) [75], and the capacitated lot sizing problem with setup times (CLSPST) [114]. These problems are known to lead to very degenerate restricted master problems, a property that usually causes instability in the standard column generation method [10, 27, 75]. We compare the performance of the primal-dual column generation technique with the standard column generation method and the analytic centre cutting plane method.

The other contributions are with regard to reoptimizing the primal-dual column generation method.

Despite two decades of successful research concerning IPMs, in particular the class corresponding to primal-dual path following methods, there is still a lot of room for improvement in reoptimization techniques. It is well known that taking advantage of the previously solved RMP could lead to important savings in solving the modified RMP. However, this still seems to be an open question for applications arising in an integer optimization context.

Currently, the warmstarting strategy proposed in [56] and further improved by heuristic procedures has been utilized in the primal-dual column generation method, showing encouraging results. Despite the fact that the approach introduced in [56] has worked well in practice, it does not guarantee full primal and dual feasibility restorations nor considers the quality of the warmstarted iterate after new columns/cuts are added. The main motivation of the design of the new warmstarting strategy presented in this thesis is to close this theoretical gap.

Under suitable assumptions, the warmstarting procedure proposed in this thesis restores primal and dual feasibilities after the addition of new columns. This method relies on solving two linear optimization problems to determine the direction which recovers primal and dual feasibilities in one step. The direction is determined so that modifications in the small components at a particular solution are not large. Additionally, the strategy provides control over the new duality gap by expanding the neighbourhood of the central path. The analysis is performed for the symmetric neighbourhood of the central path. In summary, unlike the method proposed in [56] which was based on a heuristic, the method proposed in this thesis has solid theoretical foundations.

Additionally, we perform some computational experiments and demonstrate how the proposed strategy behaves in practice. The results show that the method is comparable

to the method in [56] and, in some cases, far superior to coldstart for problems with dense columns, and is as efficient as the other methods when problems with sparse columns are considered.

In summary, this thesis makes the following contributions:

i it provides a new theoretical insight for a natural stable column generation technique based on a primal-dual interior point method and supports the findings with computational evidence;

ii it shows how primal-dual interior point methods can be efficiently combined with the column generation technique for solving relaxations of integer programming problems;

iii it presents a new warmstarting strategy embedded in the primal-dual column generation method;

iv it guarantees that after new columns are added to the RMP, the proposed strategy recovers primal and dual feasibilities in one step, keeping the warmstarting iterate inside a slightly modified neighbourhood of the central path;

v it provides extensive computational evidence comparing the new approach with a coldstart approach on some applications.

As studying column generation combined with IPMs leaves some freedom and several decisions have to be taken, we would like to state the scope of this research clearly. Firstly, we are concerned with the use of a primal-dual interior point method in a column generation framework. Although other variants of interior point method could have been considered, this class has proven to be the most efficient to date [57].

Secondly, for our computational experiments, we have selected a representative set of applications considered in several studies concerning column generation strategies and stabilization techniques [17, 75].

Thirdly, in this thesis we are not interested in obtaining optimal integer solutions, which would require the development of a branch-and-price-and-cut framework. 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. We have taken the root node as reference in all our computational experiments. However, after the branching is realized, one could still apply the methodology proposed here. Hence, by improving the efficiency of the column generation procedure, we are likely to improve the overall performance on solving the integer problem to optimality.

Fourthly, we have not studied different initializations for column generation nor designed any particular method to solve the subproblem, which correspond to the "Create the initial RMP" and "Improve the RMP" procedures in Figure 1.1, respectively. These two components are very specific to each application so we have left them out of the scope of this study since we are interested in the general efficiency of column generation and not the performance for a particular application.

Fifthly, the structure of the applications is taken into account when designing the warmstarting strategy so the conclusions should be understood in this context. Additionally, we are interested in designing a warmstarting procedure when the size of the problem is changed and, therefore, a reoptimization technique restricted to changes in the data only is not considered here.

All the original developments and computational results presented in this thesis are (partially) included in three papers:

- ☐ *New developments in the primal-dual column generation technique* jointly with Jacek Gondzio and Pedro Munari [58] (already published). The contributions, theoretical developments and computational experiments of this work are presented in Chapter 5.

- ☐ *A new warmstarting strategy for the primal-dual column generation* jointly with Jacek Gondzio [59] (submitted for publication). A thorough discussion about the findings included on this paper can be found in Chapter 6.

- ☐ *Large-scale optimization with the primal-dual column generation method* jointly with Jacek Gondzio and Pedro Munari (in preparation). A brief description of this work in progress is given in Chapter 7.

Outline

In Chapter 2 we review the fundamentals of primal-dual interior point methods. We introduce the idea of the central path and its neighbourhood and provide complexity and convergence results for a primal-dual interior point algorithm that keeps its iterates in a symmetric neighbourhood of the central path.

In Chapter 3 we describe the Dantzig-Wolfe decomposition principle (DWD) and column generation. We show the equivalence of DWD and Lagrangian relaxation and justify the use of DWD over linear relaxation. We also present different cases one may encounter after applying DWD and introduce some stabilization techniques for column generation.

In Chapter 4 we present three applications in the context of integer programming: the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW) and the capacitated lot-sizing problem with setup times (CLSPST). We provide a complete derivation of the master problems and the subproblems obtained for each application.

In Chapter 5 we describe the primal-dual column generation method (PDCGM) which relies on a primal-dual interior point method to obtain well-centred and sub-optimal dual solutions so a more stable approach is obtained. A thorough analysis demonstrates the convergence of the method to ensure that, although sub-optimal dual solutions are used during intermediate iterations, the method still converges. At the end of the chapter, we present computational comparisons of the proposed method with the standard column generation method and the analytic centre cutting plane method.

In Chapter 6 we introduce our specialized warmstarting idea, providing a theoretical analysis which demonstrates how the algorithm deals with infeasibilities and the proximity to the central path. Computational experiments for solving the root node of CSP and VRPTW after applying DWD, comparing the proposed strategy with coldstart and a partial feasibility restoration technique [56], are presented.

In Chapter 7 we summarize the main outcomes of this study and discuss further developments and avenues for related research.

Chapter 2

Primal-Dual Interior Point Methods

Interior point methods have been widely used in the last two decades and the area has grown so much that it is now a very well-established methodology to solve general optimization problems using second order information. The literature about them is ample, in particular for the linear and quadratic cases [57, 63, 64, 108, 121, 122].

In this chapter we introduce the notation used in this thesis and the fundamental ideas behind primal-dual interior point methods. Additionally, we motivate the use of a neighbourhood in the feasible interior in which the iterates should remain and present convergence and complexity results for the feasible long step path-following algorithm for that specific neighbourhood.

Since we are interested in column generation used to solve linear programming problems, we will present the associated barrier problem and derive the KKT-like system of equations for the linear case. For a more general case (convex quadratic programming), we refer the reader to [57]. For general theory regarding primal-dual interior point methods, we refer the reader to Wright's book [121].

Let us consider the following primal-dual linear programming pair

$$\mathcal{P}_0 := \min \quad c^T x, \quad \text{s.t.} \quad Ax = b, \quad x \geq 0, \tag{2.1a}$$

$$\mathcal{D}_0 := \max \quad b^T y, \quad \text{s.t.} \quad A^T y + s = c, \quad s \geq 0, \tag{2.1b}$$

where $x \in \mathbb{R}^n$ is the vector of primal variables, $y \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$ are the vectors of dual and dual slack variables, respectively. $A \in \mathbb{R}^{m \times n}$ represents the coefficient matrix, where $rank(A) = m \leq n$, and $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are vectors of parameters.

In a feasible primal-dual interior point method, we aim to solve the primal and dual problems simultaneously by relying on interior solutions of the feasible set. Therefore, instead of imposing $x \geq 0$ or $s \geq 0$, we associate a logarithmic barrier term ($-\sum_{i=1}^n \ln x_i$, in the primal space or $\sum_{i=1}^n \ln s_i$, in the dual space) to replace these sets of constraints. The resulting problems are called the barrier subproblems described by

$$\mathcal{P}_\mu := \min \quad c^T x - \mu \sum_{i=1}^{n} \ln x_i, \quad \text{s.t.} \quad Ax = b, \tag{2.2a}$$

$$\mathcal{D}_\mu := \max \quad b^T y + \mu \sum_{i=1}^{n} \ln s_i, \quad \text{s.t.} \quad A^T y + s = c, \tag{2.2b}$$

where the barrier parameter μ is the penalty associated with the barrier term. Large values of μ emphasise the contribution of the second term in the objective function of the barrier subproblems and therefore, the solution is expected to be away from the boundaries and in the interior of the positive orthant. On the other hand, when μ is small, the first term takes importance and the primal-dual solution obtained by solving (2.2) approximates the optimal solution of the primal-dual pair (2.1).

One could use either the primal or dual barrier problem to derive the first order optimality conditions.

By taking the primal barrier problem (2.2a) and using duality theory [14], we get the following Lagrangian function

$$L_{\mathcal{P}}(x, y) = c^T x - \mu \sum_{i=1}^{n} \ln x_i - y^T (Ax - b), \tag{2.3}$$

where y is the (free) dual associated with constraint $Ax = b$. Applying first order optimality conditions, we get

$$\nabla_x L_{\mathcal{P}}(x, y) = c - \mu X^{-1} e - A^T y = 0, \tag{2.4}$$

where e is an n-dimensional column vector of ones and $X^{-1} = \text{diag}\{x_1^{-1}, x_2^{-1}, \ldots, x_n^{-1}\}$. Additionally, by using the following substitution, $s = \mu X^{-1} e$, the following μ-perturbed KKT system [16, 101] associated with the primal and dual barrier problems is obtained

$$\begin{aligned} Ax &= b && \text{(2.5a)} \\ A^T y + s &= c && \text{(2.5b)} \\ XSe &= \mu e, && \text{(2.5c)} \\ (x, s) &> 0, && \text{(2.5d)} \end{aligned}$$

where $S = \text{diag}\{s_1, s_2, ..., s_n\}$. The set of solutions of this system of equations for $\mu \in (0, +\infty)$ defines the central path. Equations (2.5a) and (2.5b) are the linear constraints corresponding to primal and dual feasibility, respectively, and equations (2.5c) are the perturbed complementarity conditions, which are mildly non-linear. Equations (2.5d) ensure that the primal and dual slack variables are in the positive orthants.

The same system of equations (2.5) can be obtained by associating x as the dual of constraint $A^T y + s = c$ in the dual barrier problem (2.2b), writing the Lagrangian function as

$$L_{\mathcal{D}}(y, s, x) = b^T y + \mu \sum_{i=1}^{n} \ln s_i - x^T (A^T y + s - c), \tag{2.6}$$

and using the first order optimality conditions

$$\begin{aligned} \nabla_y L_{\mathcal{D}}(y,s,x) &= 0, \\ \nabla_s L_{\mathcal{D}}(y,s,x) &= 0. \end{aligned}$$

In a primal-dual interior point method, an approximate solution to the perturbed KKT system (2.5) is obtained at each iteration by using a (damped) step of a Newton-like system of equations. The step direction is defined by the vector $(\Delta x, \Delta y, \Delta s)$ which is obtained by solving the following system of equations

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} \xi_b \\ \xi_c \\ \tau\mu e - XSe \end{bmatrix}, \tag{2.7}$$

where the primal-dual solution is (x, y, s), and $\xi_b = b - Ax$ and $\xi_c = c - A^T y - s$ are the primal and dual residuals, respectively and $\tau \in (0, 1)$ is the centering parameter. It has to be noticed that inputs (x, y, s) and therefore, the residuals ξ_b and ξ_c and μ are iteration-dependent. Also, τ can vary from one iteration to another. For instance, if $\tau = 1$, the system (2.7) defines a *centering direction* while if $\tau = 0$, we refer to it as an *affine-scaling direction*. Most algorithms choose an intermediate value so centrality is improved as well as achieving a reduction of μ (optimality measure) [121].

Note that solving (2.7) directly can be very inefficient since it requires an invertible representation of the Jacobian matrix (matrix multiplying the vector of primal and dual directions) and therefore, one usually relies on the *augmented system of equations* or the *normal equations* [121]. By noting $\xi_\mu = \tau\mu e - XSe$, the augmented system of equations can be written as

$$\begin{bmatrix} -\Theta^{-1} & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \xi_c - X^{-1}\xi_\mu \\ \xi_b \end{bmatrix} = \begin{bmatrix} \hat{\xi}_c \\ \xi_b \end{bmatrix}, \tag{2.8}$$

where $\Theta = XS^{-1}$. By solving (2.8) one can then easily determine the direction for the dual slack components as $\Delta s = X^{-1}(\xi_\mu - S\Delta x)$. Note that linear system (2.8) is symmetric but indefinite. Also, one could go one step further and work out the normal equations, by setting $\Delta x = \Theta\hat{\xi}_c - \Theta A^T \Delta y$, so the system (2.8) becomes

$$A\Theta A^T \Delta y = \xi_b + A\Theta\hat{\xi}_c. \tag{2.9}$$

The resulting system (2.9) requires the inverse of a positive definite matrix $A\Theta A^T$ where Cholesky factorization is applicable and commonly used to speed up the calculations [33]. As pointed out in [57], the normal equations are preferred over augmented system for most linear programs.

Once the direction $(\Delta x, \Delta y, \Delta s)$ is determined from (2.7), and since the next iterate is defined as $(x^+, y^+, s^+) = (x + \alpha_P \Delta x, y + \alpha_D \Delta y, s + \alpha_D \Delta s)$, suitable values for $\alpha_P \in [0, 1]$ and $\alpha_D \in [0, 1]$ are calculated bearing in mind that $(x^+, s^+) > 0$. One could be even more aggressive by allowing larger steps as shown in [21]. However, one of the key practical requirements for the fast convergence of primal-dual interior point methods is to reduce uniformly the magnitude of all complementarity products. By doing this, we aim to keep the iterates away from the boundaries and at the same time well centred in the feasible region so large steps in the directions obtained by solving system (2.7) can be taken. To achieve this, extra conditions (extra steps) may be imposed so the new iterate, more specifically, the complementarity products of the

new iterate, remain within a close distance with respect to the reference parameter μ.

After finding suitable step sizes and updating the current iterate, μ^+ is updated (usually reduced) and the primal-dual method continues iterating until a relative duality gap drops below a prescribed optimality tolerance ε.

The stopping criterion is usually defined as

$$\frac{c^T x - b^T y}{1 + |c^T x|} \leq \varepsilon. \tag{2.10}$$

One of the most remarkable characteristics of primal-dual methods is that the distance to optimality, called the duality gap, can be calculated if some conditions hold. In particular, if (2.5a) and (2.5b) are satisfied simultaneously, we have perfect information on how far the current iterate is from the optimal solution. This feature is one of the most attractive characteristics which is heavily exploited by the primal-dual column generation method (see Chapter 5 for further details).

If the point (x, y, s) is feasible in the primal-dual spaces, the duality gap can be calculated as

$$c^T x - b^T y = c^T x - (Ax)^T y = x^T (c - A^T y) = x^T s = n\mu. \tag{2.11}$$

It is clear from (2.11), that μ is the average complementarity product (*i.e.*, $\mu = x^T s/n$).

2.1 Neighbourhoods and path-following methods

For $\tau = 1$, solving system (2.7) for a particular solution (x, y, s) depends on the value of μ. We define the μ-centre, $(x(\mu), y(\mu), s(\mu))$ as the unique point satisfying conditions (2.5a)-(2.5c) and $(x(\mu), s(\mu)) > 0$ for a particular $\mu > 0$.

Let us define the primal-dual feasible and strictly feasible sets as follows

$$\mathcal{F} = \{(x, y, s) : Ax = b, A^T y + s = c, (x, s) \geq 0\},$$
$$\mathcal{F}^0 = \{(x, y, s) : Ax = b, A^T y + s = c, (x, s) > 0\}.$$

It is easy to see that $(x(\mu), y(\mu), s(\mu)) \in \mathcal{F}^0$. Uniqueness can be shown in the following theorem which is a variation of the results obtained in [89, 91].

Theorem 2.1. *Assume that $\mathcal{F}_\mathcal{P} = \{x : Ax = b\}$ is bounded and that there exists $x \in \mathcal{F}_\mathcal{P}$ such that $x > 0$. Then, for any $v \in \mathbb{R}^n_+$, there exists a unique solution (x, y, s) that satisfies*

$$Ax = b \tag{2.12a}$$
$$A^T y + s = c \tag{2.12b}$$
$$XSe = v, \tag{2.12c}$$
$$(x, s) \geq 0, \tag{2.12d}$$

Proof. Consider the following primal barrier problem

$$\min \quad c^T x - \sum_{i=1}^{n} v_i \ln x_i, \quad \text{s.t.} \quad Ax = b. \tag{2.13}$$

By assumption, there exists a strictly positive $x \in \mathcal{F}_{\mathcal{P}}$ so the problem (2.13) has a feasible solution. Moreover, the objective function of the barrier problem is strictly convex for any $v > 0$. Since matrix A is full rank, there exists a unique solution $x \in \mathcal{F}_{\mathcal{P}}$ that minimizes the objective function in (2.13). Also, there exists a unique $y \in \mathbb{R}^m$ such that

$$c - VX^{-1}e - A^T y \quad = \quad 0, \tag{2.14}$$

where $V = \text{diag}\{v_1, v_2, \ldots, v_n\}$. The uniqueness of y follows since the rank of A equals the number of variables y. Additionally, by considering the usual transformation $s = VX^{-1}e$, s is also unique. Finally, having $v > 0$ and $x > 0$, implies $s > 0$, which completes the proof. □

Having shown that for every $\mu > 0$ there exists a unique solution $(x(\mu), y(\mu), s(\mu))$, we can introduce the notion of *central path* or *central trajectory* which is defined as

$$\mathcal{C} = \{(x(\mu), y(\mu), s(\mu)) : \mu > 0\}. \tag{2.15}$$

The central path guides the algorithm to the optimal solution of the primal-dual pair (2.1) as $\mu \to 0$ while keeping (x, s) in the positive orthant and decreasing the complementarity products to zero at "almost" the same rate. The central path is the target of any path-following method. However, it is unlikely that the new point lies exactly on the central path since the step size (α_P, α_D) is usually shrunk to avoid negative solutions in some of the components of the primal and dual slack variables.

While an infeasible primal-dual algorithm allows ξ_b and ξ_c to take any value, their feasible counterpart requires $\xi_b = 0$ and $\xi_c = 0$. For all the theoretical analysis in this study, we will use a feasible path-following method.

As mentioned before, the success of primal-dual interior point methods relies on keeping the points close to the central path. In order to do so, we can require the iterates to remain in the vicinity of $\mathcal{C}$. This vicinity is known as the neighbourhood of the central path. The most well-known neighbourhoods are the wide (also known as the one-sided) neighbourhood defined by

$$\mathcal{N}_{-\infty}(\gamma) = \{(x, y, s) \in \mathcal{F}^0 : x_j s_j \geq \gamma\mu, \forall j = 1, 2, ..., n\},$$

for some $\gamma \in (0, 1)$, and the narrow (so-called 2-norm) neighbourhood defined by

$$\mathcal{N}_2(\theta) = \{(x, y, s) \in \mathcal{F}^0 : \|XSe - \mu e\|_2 \leq \theta\mu\},$$

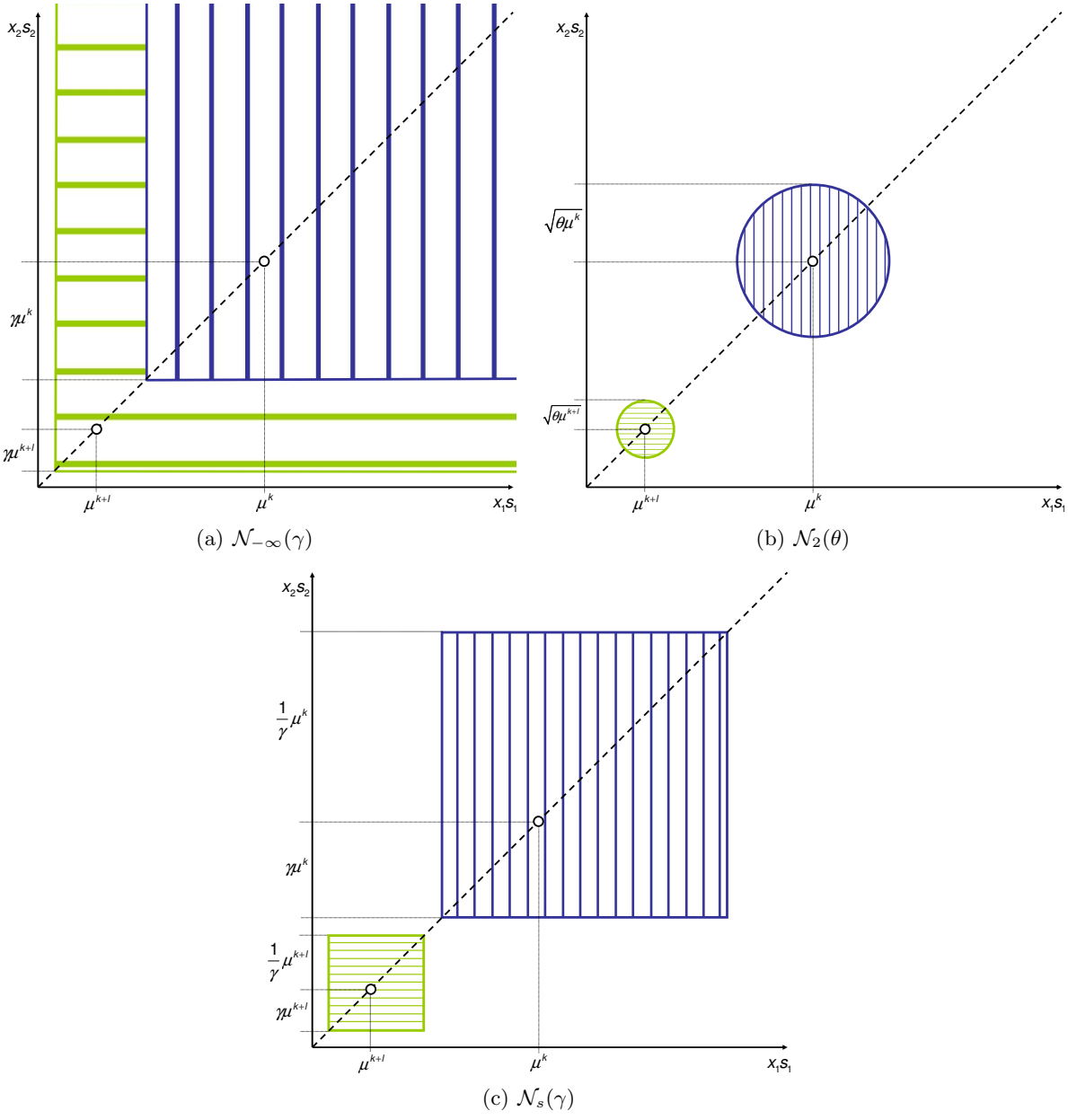
for some $\theta \in (0, 1)$. Note that the idea behind the wide neighbourhood is to keep the pair-wise complementarity products bounded from below with respect to the old barrier parameter. With the narrow neighbourhood the difference of these products and the barrier parameter is bounded from above and below and circumscribed on a n-dimensional sphere of radius $\sqrt{\theta\mu}$. As pointed out in [121], the $\mathcal{N}_{-\infty}(\gamma)$ neighbourhood is more flexible than $\mathcal{N}_2(\theta)$ due to the arbitrary choice we can make for γ. In the extreme case of γ being very close to 0, the neighbourhood resembles $\mathcal{F}^0$. For the $\mathcal{N}_2(\theta)$ case, even by choosing $\theta = 1$, the complementarity products cannot vary by more than $[0, 2\mu]$.

In addition to these two neighbourhoods of the central trajectory, in [21, 55], the

following neighbourhood is used

$$\mathcal{N}_s(\gamma) = \{(x, y, s) \in \mathcal{F}^0 : \gamma\mu \leq x_j s_j \leq \frac{1}{\gamma}\mu, \forall j = 1, 2, ..., n\}. \tag{2.16}$$

This neighbourhood controls the pair-wise complementarity products from below but also (unlike the $\mathcal{N}_{-\infty}$ neighbourhood), from above. Therefore, one may expect better practical performance by not being as conservative as with the $\mathcal{N}_2(\theta)$ neighbourhood. The drawback of this neighbourhood is that full steps are no longer ensured, worsening the theoretical complexity results of the method.



(a) $\mathcal{N}_{-\infty}(\gamma)$

(b) $\mathcal{N}_2(\theta)$

(c) $\mathcal{N}_s(\gamma)$

Figure 2.1: Neighbourhoods of the central path

As an illustrative example, let us generate two example values for μ with $n = 2$. In Figure 2.1, we have plotted the three described neighbourhoods, namely $\mathcal{N}_{-\infty}(\gamma), \mathcal{N}_2(\theta)$

and $\mathcal{N}_s(\gamma)$ in the space of the complementarity products. The diagonal dotted line defines the central trajectory while the areas with horizontal and vertical lines represent the corresponding neighbourhoods associated with a given μ (denoted by a circle). The analysis of these figures, hints at why the path-following methods are classified into the ones which use long or short steps.

Since the symmetric neighbourhood $\mathcal{N}_s(\gamma)$ will be used extensively in the warm-starting analysis in Chapter 6, in the following section we recall the convergence and complexity results for a long-step path following algorithm using the symmetric neighbourhood.

2.2 Convergence and complexity of the $\mathcal{N}_s(\gamma)$ neighbourhood

In this section we summarize the convergence and complexity results for the symmetric neighbourhoods presented in [20]. The analysis closely follows the results for the long-step path following method presented in [121, Chapter 5]. For the sake of completeness, we also provide proofs of some lemmas which were missing in [20, 21, 121].

2.2.1 Algorithm

Firstly, let us consider a feasible primal-dual interior point method and therefore $\xi_b = \xi_c = 0$ in system (2.7). Algorithm 2.1 is a variation of the long-step path following method introduced in [121, Chapter 5] but in our case we use the symmetric neighbourhood.

Algorithm 2.1 Long-Step Path Following Algorithm with Symmetric Neighbourhood

Input: $\gamma, \tau_{min}, \tau_{max}$ with $\gamma \in (0,1), 0 < \tau_{min} < \tau_{max} < 1$ and $(x^0, \lambda^0, s^0) \in \mathcal{N}_s(\gamma)$;

1: **for** $k = 0, 1, 2, ...$ **do**
2: choose $\tau^k \in [\tau_{min}, \tau_{max}]$;
3: solve

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S^k & 0 & X^k \end{bmatrix} \begin{bmatrix} \Delta x^k \\ \Delta y^k \\ \Delta s^k \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \tau^k \mu^k e - X^k S^k e \end{bmatrix} \tag{2.17}$$

to obtain $(\Delta x^k, \Delta y^k, \Delta s^k)$;
4: choose α_k as the largest value of $\alpha \in [0,1]$ such that $(x^k(\alpha), y^k(\alpha), s^k(\alpha)) \in \mathcal{N}_s(\gamma)$;
5: set $(x^{k+1}, y^{k+1}, s^{k+1}) = (x^k(\alpha_k), y^k(\alpha_k), s^k(\alpha_k)) \in \mathcal{N}_s(\gamma)$;
6: **end for**

2.2.2 Convergence

The following lemma provides a bound on $\|\Delta X \Delta S e\|$. This technical result is the same as the one obtained in [121, Lemma 5.10] and the proof can be found in that textbook.

Lemma 2.2. *If* $(x, y, s) \in \mathcal{N}_s(\gamma), then$

$$\|\Delta X \Delta S e\| \leq 2^{-3/2}(1 + \frac{1}{\gamma})n\mu.$$

The next lemma ensures that feasibility is kept by the algorithm if the starting point is feasible.

Lemma 2.3. *Let the step* $(\Delta x, \Delta y, \Delta s)$ *be defined by*

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \tau\mu e - XSe \end{bmatrix}. \tag{2.18}$$

Then

$$\Delta x^T \Delta s = 0$$

Proof. From the first and the second rows of (2.18) it follows that

$$A\Delta x = 0 \tag{2.19}$$

$$A^T \Delta y + \Delta s = 0. \tag{2.20}$$

Multiplying (2.20) by Δx^T, we obtain

$$\begin{aligned} 0 &= \Delta x^T A^T \Delta y + \Delta x^T \Delta s \\ &= (A\Delta x)^T \Delta y + \Delta x^T \Delta s. \end{aligned} \tag{2.21}$$

Replacing (2.19) in (2.21), we get

$$\Delta x^T \Delta s = \sum_{j=1}^{n} \Delta x_j \Delta s_j = 0,$$

as claimed. □

The following lemma corresponds to [20, Theorem 2.6.]. This lemma provides a bound on α so that the new iterate remains in the symmetric neighbourhood of the central trajectory.

Lemma 2.4. *If* $(x, y, s) \in \mathcal{N}_s(\gamma)$, $(x(\alpha), y(\alpha), s(\alpha)) \in \mathcal{N}_s(\gamma)$ *for all*

$$\alpha \in \left[0, \frac{2^{3/2}}{n}\gamma\frac{1-\gamma}{1+\gamma}\tau^k\right].$$

The proof can be found in Colombo's thesis [20]. Lemmas 2.2, 2.3 and 2.4 are useful when proving the convergence of the method which is stated in the following theorem.

Theorem 2.5. *Given the parameters* γ, τ_{min} *and* τ_{max} *in Algorithm 2.1, there is a constant* δ *independent of* n *such that*

$$\mu^{k+1} \leq \left(1 - \frac{\delta}{n}\right)\mu^k \tag{2.22}$$

for all $k \geq 0$.

The proof is skipped since the details can be found in [20, Theorem 2.7.].

2.2.3 Complexity

To analyse the complexity of Algorithm 2.1 we first need a technical result on logarithmic functions.

Lemma 2.6. *We have* $\log(1+\beta) \leq \beta$ *for all* $\beta > -1$*, with equality if and only if* $\beta = 0$*.*

Proof. Let us define a function

$$f(\beta) \quad = \quad \beta - \log(1+\beta).$$

Therefore, the claim can be proven if

$$f(\beta) \geq 0, \qquad \forall \beta > -1.$$

with equality (*i.e.*, $f(\beta) = 0$) if and only if $\beta = 0$. Using first and second order optimality conditions, we observe that f has a minimum at point β if

$$f'(\beta) = 1 - \frac{1}{1+\beta} = 0,$$

$$f''(\beta) = \frac{1}{(1+\beta)^2} > 0.$$

Since the function is convex its (only) minimum is attained at $\beta = 0$. Hence, $f(0) = 0$ which completes the proof. □

The complexity result of Algorithm 2.1 is an immediate consequence of Theorem 2.5, [121, Theorem 3.2.] and Lemma 2.6.

Theorem 2.7. *Given* $\epsilon > 0$ *and* $\gamma \in (0,1)$*, suppose that the starting point* $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$ *in Algorithm 2.1 satisfies*

$$\mu^0 \leq \frac{1}{\epsilon^\kappa} \tag{2.23}$$

for some positive constant κ*. Then there is an index* K *with* $K = \mathcal{O}(n \log \frac{1}{\epsilon})$ *such that*

$$\mu^k \leq \epsilon, \qquad \forall k \geq K. \tag{2.24}$$

The proof can be found in [20, Theorem 2.8.]

It is important to point out that both convergence and complexity results are the same as those obtained for a long-step path following algorithm using a $\mathcal{N}_{-\infty}(\gamma)$ neighbourhood. It is clear that when using $\mathcal{N}_s(\gamma)$ we obtain better control over each complementarity product j at each iteration k since each of them is bounded from above and from below. This remedies possible bad behaviour when the complementary products are unbalanced, a phenomenon which may occur when a one-sided neighbourhood is used. Moreover, this is achieved without losing polynomial complexity, $\mathcal{O}(n \log \frac{1}{\epsilon})$ for a predefined tolerance ϵ.

Chapter 3

Column Generation and the Decomposition Principle

In this chapter we describe column generation and the Dantzig-Wolfe decomposition principle. We first introduce Dantzig-Wolfe decomposition and show its equivalence with Lagrangian relaxation. Then, we describe the standard column generation method, point out well-known drawbacks of this version and briefly describe some of the methods proposed to remedy these difficulties.

There exists a close relation between Dantzig-Wolfe decomposition and column generation. The Dantzig-Wolfe decomposition principle is one of several ways of reformulating a problem by means of describing the feasible set fully or partially with an exponential number of variables in order to exploit the underlying structure of the problem. It has similarities with Benders decomposition [12] and an equivalence with Lagrangian relaxation [45]. On the other hand column generation is an iterative process which includes generating and adding a subset of this exponential number of variables and checking whether an optimal or sub-optimal solution has been achieved and has an equivalence with the cutting plane method [78].

3.1 Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition principle (DWD) is a technique proposed for linear programming problems with a special structure in the coefficient matrix. The original aim of this technique is to make large linear problems tractable as well as to speed up the solution by the simplex method [25]. Although the name is due to Dantzig and Wolfe for their contribution in [25], the method was inspired by the work of Ford and Fulkerson [41] where they proposed a multicommodity flow formulation where variables represent path flows for each commodity. This decomposition principle was also considered in [46] to solve the cutting stock problem and, shortly after, the method was proposed to tackle a two-stage stochastic programming problem [24]. Except for some classes of problems, DWD was not advantageous for general linear programming problems. However, it showed itself to be very successful when extended to integer programming problems [8, 116]. In this context, the focus was on providing stronger bounds when solving linear relaxations in order to speed up a branch-and-bound search. In this thesis we are interested in developing this technique and using it to solve the root node of some problems arising in combinatorial optimization.

Let us consider the following integer programming problem (IP):

$$\begin{aligned} \min \quad & c^T x, && \text{(3.1a)} \\ \text{s.t.} \quad & Ax \leq b, && \text{(3.1b)} \\ & x \in \mathcal{X}, && \text{(3.1c)} \end{aligned}$$

where $\mathcal{X} = \{x \in \mathbb{Z}^n_+ : Dx \leq d\}$ is a discrete set, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $D \in \mathbb{R}^{h \times n}$, $d \in \mathbb{R}^h$. Also, let us denote the optimal solution of problem (3.1) as z_{IP}. Let us assume that without the presence of constraint (3.1b), the problem would be easily solved by taking advantage of the structure of $\mathcal{X}$, particularly of matrix D (see Figure 3.1).

The Dantzig-Wolfe decomposition principle can be applied to integer programming formulations that have a special structure in the coefficient matrix. Usually, the matrix is very sparse and composed of several blocks which are almost independent of each other except for the existence of a set of linking constraints. By linking set of constraints we mean the set of constraints that couple several variables. In Figure 3.1, we illustrate a possible matrix structure of problem (3.1) known as *block diagonal*. This is usually referred to as the set of global constraints. For instance, matrix A can describe the company's yearly budget while the decomposable part, matrices $D^1, D^2, \ldots, D^K$, may represent subsidiary conditions.

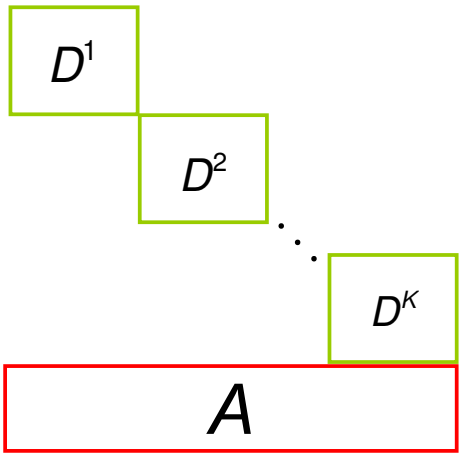


Figure 3.1: Block diagonal structure

Note that even though block structure is desirable from a decomposition and a parallel computation point of view, it is not a requirement. The desirable property is that the solution of the problem without the linking constraints can be calculated more efficiently than the solution with their presence.

To apply DWD to the integer programming problem (3.1), we consider the convexification approach [115], although an alternative approach could have been used (see [116]). Consider the convex hull of the set $\mathcal{X}$, denoted by $\mathcal{C} = conv(\mathcal{X})$. Now, assume that we know the sets of all extreme points p_q and extreme rays p_r that fully represent $\mathcal{C}$. Hence, we can write any $x \in \mathcal{C}$ as

$$x = \sum_{q \in \mathcal{Q}} \lambda_q p_q + \sum_{r \in \mathcal{R}} \mu_r p_r, \text{ with } \sum_{q \in \mathcal{Q}} \lambda_q = 1,$$

where the sets $\mathcal{Q}$ and $\mathcal{R}$ consist of indices of all extreme points and extreme rays of $\mathcal{C}$, respectively. By using this equality in problem (3.1), we obtain the equivalent

formulation

$$\min \quad \sum_{q\in\mathcal{Q}} \lambda_q (c^T p_q) + \sum_{r\in\mathcal{R}} \mu_r (c^T p_r), \tag{3.2a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} \lambda_q (Ap_q) + \sum_{r\in\mathcal{R}} \mu_r (Ap_r) \leq b, \tag{3.2b}$$

$$\sum_{q\in\mathcal{Q}} \lambda_q = 1, \tag{3.2c}$$

$$\lambda_q \geq 0,\ \mu_r \geq 0, \qquad \forall q \in \mathcal{Q}, \forall r \in \mathcal{R}, \tag{3.2d}$$

$$x = \sum_{q\in\mathcal{Q}} \lambda_q p_q + \sum_{r\in\mathcal{R}} \mu_r p_r, \tag{3.2e}$$

$$x \in \mathbb{Z}^n_+. \tag{3.2f}$$

Notice that we still need to keep $x \in \mathbb{Z}^n_+$ in order to guarantee the equivalence between (3.2) and (3.1). However, relaxing the integrality of x in (3.2) usually leads to a lower bound that is the same as or stronger than the one obtained by the linear programming relaxation of (3.1) (this will be shown in Section 3.3). For this reason, the relaxation of (3.2) can lead to improvements when solving (3.1) by a branch-and-bound approach. This is because a tighter relaxation is obtained and therefore the space search for the branch-and-bound algorithm is reduced.

Assume we have relaxed the integrality condition on x (linear relaxation). Thus, there is no need to keep the constraints (3.2e) in the formulation of (3.2). By denoting $c_q = c^T p_q$, $a_q = Ap_q$, $\forall q \in \mathcal{Q}$ and $c_r = c^T p_r$, $a_r = Ap_r$, $\forall r \in \mathcal{R}$, a *relaxation* of the problem (3.2) is given by

$$\min \quad \sum_{q\in\mathcal{Q}} c_q \lambda_q + \sum_{r\in\mathcal{R}} c_r \mu_r, \tag{3.3a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} a_q \lambda_q + \sum_{r\in\mathcal{R}} a_r \mu_r \leq b, \tag{3.3b}$$

$$\sum_{q\in\mathcal{Q}} \lambda_q = 1, \tag{3.3c}$$

$$\lambda_q \geq 0,\ \mu_r \geq 0, \qquad \forall q \in \mathcal{Q}, \forall r \in \mathcal{R}, \tag{3.3d}$$

which is called the Dantzig-Wolfe master problem (DW-MP). Let us denote z_{DWD} as the optimal solution of (3.3). The dual problem associated with (3.3) has the same form as the problem obtained after applying Lagrangian relaxation for integer programming in the original problem (3.1) [45, 86, 106]. The equivalence between both techniques will be discussed in Section 3.2.

As depicted in Figure 3.1, for some classes of problems, 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

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

where $|L_k|$ is the number of variables associated with $\mathcal{X}_k$, and x^k is the vector containing the components of x associated with $\mathcal{X}_k$. Note that the number of variables associated with every set does not have to be the same and varies with the problems we are dealing with. For simplicity, we assume that the set $\mathcal{X}$ is bounded and, hence, $\mathcal{R} = \emptyset$, although

the following discussion can be extended to deal with unbounded cases (see [100]).

When we can decompose matrix D, DW-MP can be rewritten as

$$\min \quad \sum_{k=1}^{K} \sum_{q \in \mathcal{Q}_k} c_q^k \lambda_q^k, \tag{3.4a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} \sum_{q \in \mathcal{Q}_k} a_q^k \lambda_q^k \leq b, \tag{3.4b}$$

$$\sum_{q \in \mathcal{Q}_k} \lambda_q^k = 1, \qquad \forall k = 1, \ldots, K, \tag{3.4c}$$

$$\lambda_q^k \geq 0, \qquad \forall q \in \mathcal{Q}_k,\ \forall k = 1, \ldots, K, \tag{3.4d}$$

where the extreme points of the subset $\mathcal{X}_k$ are represented by each p_q with $q \in \mathcal{Q}_k$. This form is know as the *disaggregated* master problem. In the case where D is not decomposable, namely when $\mathcal{X} = \{x \in \mathbb{Z}_+^n : Dx \leq d\}$ and $\mathcal{R} = \emptyset$, we can rewrite (3.3) as the following *aggregated* master problem

$$\min \quad \sum_{q \in \mathcal{Q}} c_q \lambda_q, \tag{3.5a}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{Q}} a_q \lambda_q \leq b, \tag{3.5b}$$

$$\sum_{q \in \mathcal{Q}} \lambda_q = 1, \tag{3.5c}$$

$$\lambda_q \geq 0, \qquad \forall q \in \mathcal{Q}, \tag{3.5d}$$

where $q \in \mathcal{Q}$ and $\mathcal{Q}$ represents the set of extreme points of $\mathcal{C}$. Note that the disaggregated version has more constraints than the aggregated one. However, the number of extreme points to represent $\mathcal{C}$ is larger in the aggregated version. Despite the fact that technically both problems offer the same optimal solution, the way of generating the columns may differ, and a disaggregated form is usually preferred over its aggregated counterpart.

3.2 Lagrangian relaxation equivalence

As stated in [86, 106], there is a strong relation between the bounds obtained applying DWD and Lagrangian relaxation [39, 45]. This is a well-known result that we state here for the sake of completeness.

Recalling the original problem (3.1), let us associate $u \in \mathbb{R}_-^m$ as the Lagrangian multiplier for constraints (3.1b). Then, the partial Lagrangian function becomes

$$L(x, u) = c^T x - u^T (Ax - b) = b^T u + (c^T - u^T A)x.$$

Note that for an arbitrary non-positive value of u, the problem

$$\begin{aligned} L(u) &= \min_{x \in \mathcal{X}} L(x, u), \\ &= b^T u + \min_{x \in \mathcal{X}} (c^T - u^T A)x, \\ &= b^T u + \min_{x \in \mathcal{C}} (c^T - u^T A)x, \end{aligned}$$

provides a lower bound for the optimal solution of the primal problem, z_{IP}. Since we are interested in the best lower bound, we are interested in finding

$$\mathcal{L} := \max_{u \in \mathbb{R}^m_-} L(u). \tag{3.6}$$

Using the notation p_q with $q \in \mathcal{Q}$ and p_r with $r \in \mathcal{R}$ as the extreme points and extreme rays introduced earlier, which describe $\mathcal{C}$, we get the following Lagrangian bounds:

(i) $L(u) = -\infty$, if $(c^T - u^T A)p_r < 0$ for some $r \in \mathcal{R}$; and

(ii) $L(u) = b^T u + (c^T - u^T A)p_q$, for some $q \in \mathcal{Q}$, otherwise.

Since we have assumed that the master problem (3.3) has a finite optimal solution, we can impose $L(u) > -\infty$. To avoid such solution, we include a set of constraints to prevent optimal unbounded directions. Then, the Lagrangian dual problem (3.6) can be rewritten as the following linear programming problem

$$\begin{aligned}
\max \quad & v, && && (3.7a)\\
\text{s.t.} \quad & u^T A p_r \leq c^T p_r, && \forall r \in \mathcal{R}, && (3.7b)\\
& v + u^T A p_q - b^T u \leq c^T p_q, && \forall q \in \mathcal{Q}, && (3.7c)\\
& u \leq 0. && && (3.7d)
\end{aligned}$$

Constraints (3.7b) guarantee $(c^T - u^T A)p_r \geq 0$ for every $r \in \mathcal{R}$ avoiding a Lagrangian bound of type (i). Additionally, introducing variable $v \in \mathbb{R}$, the Lagrangian bound of type (ii) is obtained by maximizing v and considering constraints (3.7c). Using the same transformation as in the previous section, namely $c_j = c^T p_j$ and $a_j = Ap_j$, for $j \in \mathcal{Q} \cup \mathcal{R}$, we get the following equivalent problem

$$\begin{aligned}
\max \quad & v, && && (3.8a)\\
\text{s.t.} \quad & a_r^T u \leq c_r, && \forall r \in \mathcal{R}, && (3.8b)\\
& v + a_q^T u - b^T u \leq c_q, && \forall q \in \mathcal{Q}, && (3.8c)\\
& u \leq 0. && && (3.8d)
\end{aligned}$$

It is easy to see that by associating dual variables $\mu_r \geq 0$ for every $r \in \mathcal{R}$ and $\lambda_q \geq 0$ for every $q \in \mathcal{Q}$ with constraints (3.8b) and (3.8c), respectively, and writing the dual of problem (3.8), we get (3.3). Therefore, due to duality theory, we can conclude that the bound obtained by solving (3.8) is the same as the one obtained by solving (3.3). A different way of showing this equivalence is to associate $u \in \mathbb{R}^m_-$ and $v \in \mathbb{R}$ as dual variables of constraints (3.3b) and (3.3c), respectively and derive the dual problem associated with (3.3). The resulting problem is (3.8).

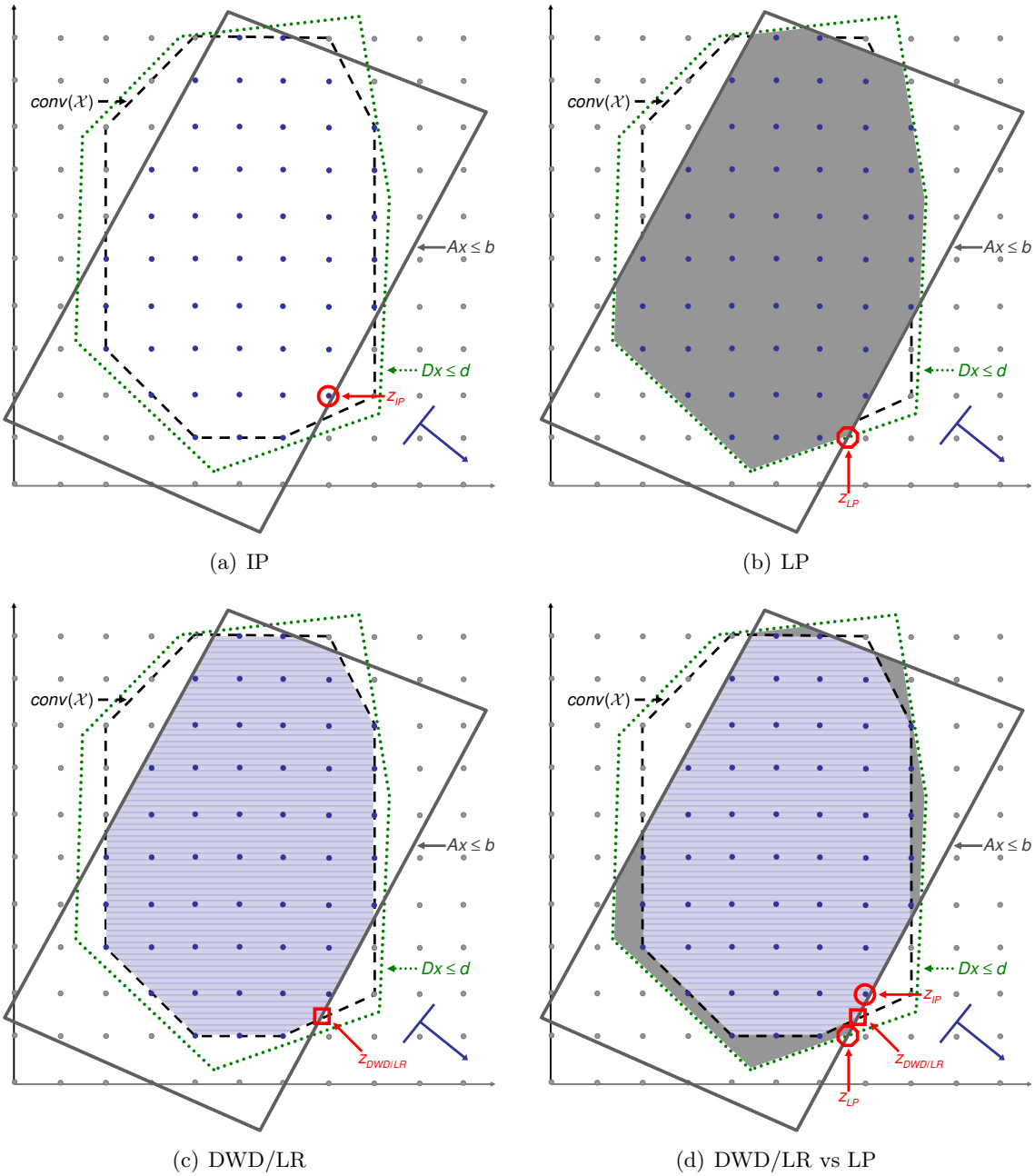
3.3 A graphical interpretation

As pointed out at the beginning of this chapter, the main idea of using the Dantzig-Wolfe decomposition principle is to get a better lower bound for a branch-and-bound tree than if linear relaxation is applied to the original problem. A well-known result [45, Theorem 1] is partially restated here in a simplified form suiting our interests.

Theorem 3.1. *Having a problem of the form* (3.1) *with at least one feasible solution, the following bound relations are always valid*

$$z_{IP} \geq z_{DWD} = z_{LR} \geq z_{LP}, \tag{3.9}$$

where z_{IP} *represents the optimal solution of the integer program* (3.1), z_{DWD} *is the optimal solution of* (3.3), z_{LR} *the optimal solution of* (3.8), *and* z_{LP} *denotes the optimal solution of a linear relaxation to* (3.1).



(a) IP (b) LP

(c) DWD/LR (d) DWD/LR vs LP

Figure 3.2: Feasible solutions for the IP, DWD/LR and LP formulations

A proof of this theorem can be found in [45]. In this section, we are interested in giving a more intuitive interpretation of these results using some graphical aid. To do

so, let us consider Figure 3.2 which illustrates feasible and optimal solutions for the aforementioned problems. In all the subfigures the contour of the feasible polytope described by constraints (3.1b) ($Ax \leq b$) is represented by a solid line. Also, we use a dotted line to illustrate the shape of the polytope formed by constraints $Dx \leq d$. The convex hull of $\mathcal{X}$ is represented by a dashed line and the direction in which the objective function is minimized is indicated by the arrow at the bottom right corner. Subfigure 3.2(a) represents the feasible set of the original problem (3.1) considering integer solutions. In Subfigure 3.2(b) the feasible region after applying linear relaxation to (3.1) (LP) is shown. From this subfigure, it is clear that the solution one can obtain by solving z_{LP} is a lower bound on z_{IP} since we do not force integer solutions and that the feasible solutions of z_{IP} are included in the feasible polytope considered in z_{LP}. In contrast, Subfigure 3.2(c), displays the corresponding primal (dual) feasible region after applying DWD (LR) and relaxing integrality on x. It is important to notice that $conv(\mathcal{X})$ is very likely to be different from the feasible region illustrated in Subfigure 3.2(b). Observe that similar to the previous case, the feasible set of z_{IP} is included in the polytope obtained by using the z_{DWD}/z_{LR} approach and therefore, the bounds obtained by any of these reformulations satisfy $z_{IP} \geq z_{LP}$ and $z_{IP} \geq z_{DWD} = z_{LR}$. Finally, in Subfigure 3.2(d) the gains obtained by applying DWD/LR over LP are illustrated. We have overlapped the feasible region of Subfigure 3.2(c) over Subfigure 3.2(b), where the shaded region is the feasible region removed from the LP by using DWD/LR instead. This shows that the feasible set of DWD/LR is contained in the LP feasible polytope. Hence, we have $z_{DWD} = z_{LR} \geq z_{LP}$. In general, by using DWD [25], one should expect to get a tighter lower bound for the original problem than by applying a linear relaxation.

3.4 Column generation

Decomposing an integer problem with DWD results in a linear programming problem which has an exponential number of variables. Moreover, in many situations the variables are not even explicitly available and have to be generated by calling a specialized procedure. According to [99], the first reference to the term *column generation* was made in [3]. However, the method was proposed a few years earlier by Gilmore and Gomory [46, 47]. In those papers from the early 60's, Gilmore and Gomory suggested solving the *cutting stock* problem by using cutting patterns as variables and generating them by solving a *knapsack* problem. To motivate this section, we briefly describe the one-dimensional cutting stock problem. A more formal definition is given in Chapter 4. The cutting stock problem deals with finding the cutting patterns which minimize the waste of material such that demands for pieces of different widths are satisfied. In the one-dimensional case, one can think of paper rolls that need to be cut into smaller pieces to satisfy some customer's demands. Considering a specific cutting pattern as a vector of integer values, where every value represents the number of pieces of a particular size to be cut, one may end up with an exponential number of columns. This number becomes very large when the stock width is large and the demanded pieces are of smaller widths. For real size instances, the direct approach of listing all the possible cutting patterns becomes impractical and therefore one should avoid it. A way of solving this problem (and many more with an exponential number of variables) is to start with only a small set of columns and add more only if needed. This whole process is known as column generation because, from a linear algebra and simplex point of view, the objective and constraint coefficients associated with a variable come as a new column

to be appended to the working tableau. The interest in column generation is huge and has attracted the attention of the integer programming community for several decades but it has been accentuated during the past 10 years due to the computational and theoretical developments in the techniques used to generate columns. Additionally, the better understanding of how to remedy the main difficulties when using the standard approach has made this technique widely known and used. Moreover, the variety of problems this can be applied to is vast. For instance, the cutting stock problem is in the core formulation of many problems due to its simple structure (*e.g.*, *air crew scheduling* [7, 65] and *bin packing* [18, 48, 84, 85]) and therefore, of interest to many researchers.

More formally, let us consider a master problem (MP) represented by the following linear programming problem

$$z^\star := \min \quad \sum_{j \in N} c_j \lambda_j, \tag{3.10a}$$

$$\text{s.t.} \quad \sum_{j \in N} a_j \lambda_j = b, \tag{3.10b}$$

$$\lambda_j \geq 0, \qquad \forall j \in N, \tag{3.10c}$$

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 $a_j \in \mathbb{R}^m$, $\forall j \in N$. The similarity with problem (3.5) is obvious. Note that the convexity constraint $\sum_{j \in N} \lambda_j = 1$, is implicitly included in constraints (3.10b).

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 procedure. For instance, for the cutting stock problem, the columns (patterns) are generated by solving the knapsack problem [46]. Hence, instead of solving the MP directly, we consider only a small subset of columns at first, which leads to a reduced version of (3.10), known as the restricted master problem (RMP):

$$z_{RMP} := \min \quad \sum_{j \in \overline{N}} c_j \lambda_j, \tag{3.11a}$$

$$\text{s.t.} \quad \sum_{j \in \overline{N}} a_j \lambda_j = b, \tag{3.11b}$$

$$\lambda_j \geq 0, \qquad \forall j \in \overline{N}, \tag{3.11c}$$

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.*, $z^\star \leq z_{RMP}$.

Column generation is an iterative process where the RMP is solved and the optimal solution is used to generate one or more new columns. Then, the RMP is modified by adding the generated column(s) and the same steps are repeated until once can guarantee that no more columns are necessary to obtain an optimal solution of the MP. In an extreme case one may end up solving an RMP which has the same number of columns as the MP. However, in practice, one ends up solving RMPs with much fewer columns than the associated MP.

Natural questions arise at this point: (a) how to check whether no more columns are necessary? and (b) how to generate new columns to be added to the RMP? The answers to both questions are given by the *oracle*. The oracle is composed of one or more (pricing) subproblems, which are able to generate new columns by using a dual solution of the RMP. The oracle's function is to check if a dual solution of the RMP is also feasible for the MP.

Let $u \in \mathbb{R}^m$ be the vector of free dual variables associated with constraints (3.10b) of the MP. The dual of (3.10) can be written as

$$\begin{aligned} \max \quad & b^T u, && && (3.12a)\\ \text{s.t.} \quad & a_j^T u + s_j = c_j, && \forall j \in N, && (3.12b)\\ & s_j \geq 0, && \forall j \in N. && (3.12c) \end{aligned}$$

For any given primal-dual solution $(\overline{\lambda}, \overline{u})$, we assume that $\overline{\lambda}$ is a primal feasible solution of (3.10). We can verify the feasibility of the dual variables in the MP by checking the sign of the reduced costs $s_j = c_j - \overline{u}^T a_j$, for each $j \in N$ (see (3.12b)). If $s_j < 0$ for some $j \in N$, then the dual solution $\overline{u}_j$ is not dual 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 calculating the value of s_j for every $j \in N$. Instead, we use the minimum among them, which is obtained by solving the subproblem

$$z_{SP}(\overline{u}) := \min\left\{0; c_j - \overline{u}^T a_j | a_j \in \mathcal{A}\right\}. \tag{3.13}$$

The value $z_{SP}(\overline{u})$ is called the *value of the oracle*. If $\overline{u}$ is the optimal dual solution for the RMP and $z_{SP}(\overline{u}) = 0$, 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. Note that more than one column may be found by the oracle and therefore, we can add one or more of them to the RMP. Actually, any column with a negative reduced cost (*i.e.,* $c_j - \overline{u}^T a_j < 0$) can be added to the RMP. By using (3.13) we can provide a lower bound of the optimal value of the MP if we know a constant κ such that

$$\kappa \geq \sum_{i \in N} \lambda_i^\star, \tag{3.14}$$

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

$$z_{RMP} + \kappa \times z_{SP}(\overline{u}) \leq z^\star \leq z_{RMP}. \tag{3.15}$$

The value of κ is usually known when DWD is applied (see constraints (3.4c) and (3.5c) for instance).

Column generation terminates when both the lower and upper bounds in (3.15) 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 and the iterations in this process are called *inner* iterations.

To summarize the above discussion, in Algorithm 3.1 we describe the standard column generation method, where δ denotes the optimality tolerance and $\overline{z}_{SP} := z_{SP}(\overline{u})$.

Algorithm 3.1 The Standard Column Generation Method

Input: Initial RMP; $\delta > 0$.

Set: LB $= -\infty$, UB $= \infty$, $gap = \infty$;

1: **while** $(gap \geq \delta)$ **do**
2: find an optimal solution $(\overline{\lambda}, \overline{u})$ of the RMP;
3: UB $= \overline{z}_{RMP}$;
4: call the oracle with the query point $\overline{u}$ and set $\overline{z}_{SP} := z_{SP}(\overline{u})$;
5: LB $= \max\{\text{LB}, \kappa\overline{z}_{SP} + UB\}$;
6: $gap = (\text{UB} - \text{LB})/(1 + |\text{UB}|)$;
7: if $(\overline{z}_{SP} < 0)$ then add the new columns to the RMP;
8: **end while**

3.4.1 Adding columns to the RMP

When setting the number of columns to be added to the RMP per iteration different situations can occur. The number of columns provided by the subproblems depends on the structure of the problem and the ability of the oracle solver to generate columns with negative reduced costs. Let us recall DWD notation but keep in mind the column generation framework.

As shown in Section 3.1, depending on the structure of the problem, we can get an aggregated or disaggregated formulation.

First let us consider the disaggregated case (3.4). In this case, K independent subproblems are obtained and, therefore, K columns can be added to the RMP per iteration. In fact, if we denote by u and v the dual variables associated with constraints (3.4b) and (3.4c), respectively, we have the following subproblem for each $k = 1, \ldots, K$

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

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

In a special case when the K subproblems are identical, they generate the same columns for a given dual point $\overline{u}$. We can take advantage of such a situation by using an aggregation of variables in the following way

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

As a consequence, we can drop the index k from $\mathcal{Q}_k$ and denote it simply by $\mathcal{Q}$, since all $\mathcal{Q}_k$ represent the same set. The same simplification may be applied to the parameters c_q^k and a_q^k. Note that even though after considering all these changes together, we can rewrite problem (3.4) as (3.5) (except for constraint (3.5c)), the subproblem obtained has reduced dimensions and solving it is usually preferred over solving the subproblem obtained when using the aggregated form. Also, notice that similar to DW-MP, there is now only one subproblem associated with the aggregated master problem, which is given by any z_{SP}^k, since they are all identical. If $0 \in \mathcal{X}_k$ and its associated cost is also zero, then the equality in constraint (3.5c) can be relaxed to

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

Besides, if K is sufficiently large, then this inequality holds strictly in the optimal solution and, hence, (3.5c) can be dropped from the problem. Since by aggregating variables we lose $\sum_{q \in Q_k} \lambda_q^k = 1$ for every $k = \{1, \ldots, K\}$, one may consider this formulation as a generalization of DWD.

A third case is when we can add many more different columns per iteration. This situation is independent of the way we decompose the problem but related to the subproblem solver. Therefore, it may happen in the aggregated and disaggregated cases. As mentioned previously, 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 p solutions, for a given $p > 0$, then we can generate up to p 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 required to obtain them.

3.4.2 Limitations of the standard version

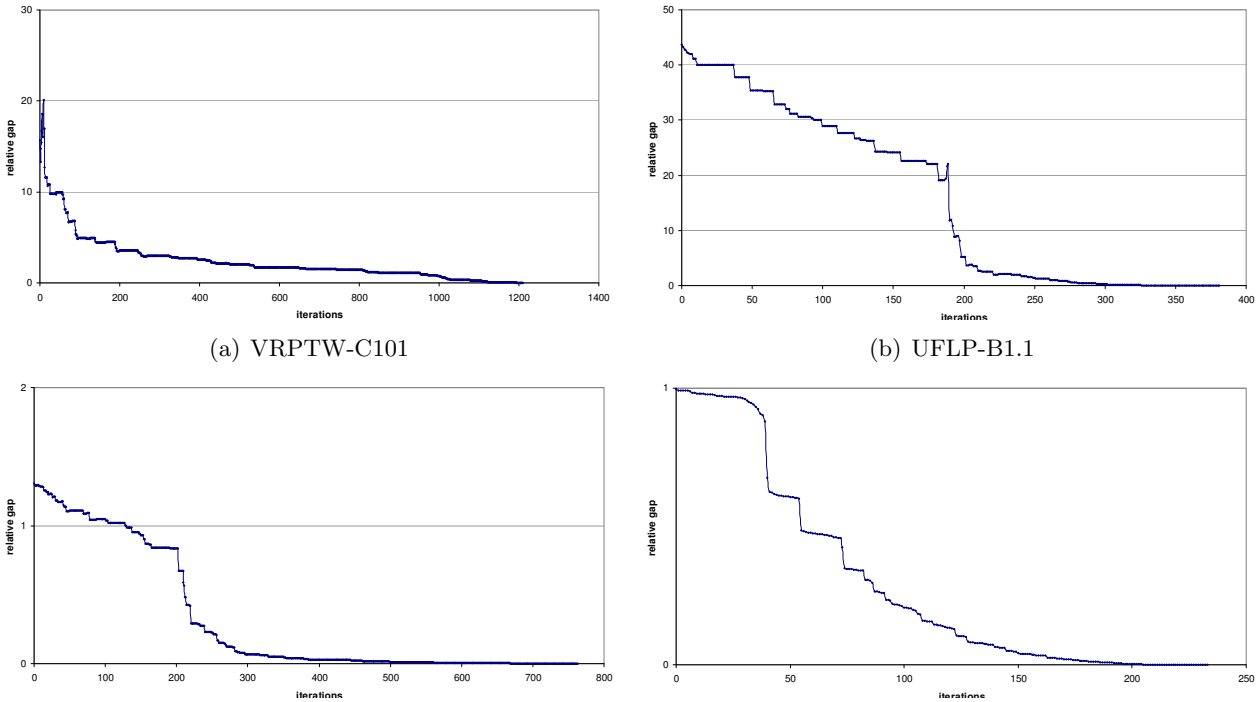
The standard version of column generation based on simplex-type methods is known to be unstable due to the use of extreme dual variables [29, 117]. Two problems usually arise when using these extreme solutions.

The first one is due to large oscillations in the value of the dual variables at the beginning of the process. This is known as the *heading-in* effect [117]. It occurs since during the first iterations of column generation only a small subset of columns is available and, therefore, the dual information provided by them and sent to the subproblems, is of limited use. This results in a considerable number of calls to the subproblem solver before gathering any useful columns. This problem is accentuated when the initialization of the method (initial columns) is poor. Moreover, it has been observed that the initial dual variables differ completely from the dual solutions obtained close to termination. As it turns out, this issue has attracted much of the attention of the column generation community and a wide variety of methods have been proposed to deal with this weakness.

The second problem when using the standard column generation method and the simplex method to obtain the solution of the RMPs, is when approaching the optimal solution of the MP. Many of the applications suffer from degeneracy and in a column generation process this results in slow convergence towards optimality known as the *tailing-off* effect. In other words, and as described in [29], an optimal solution in the original components x can be described by many convex combinations of extreme points (λ components), making the right choice difficult. Therefore, one should expect very little progress when approaching optimality. Using a graphical interpretation [47], this can be observed as a large tail if we plot the progress of the gap against the iteration counter.

In order to see how these two limitations affect the performance of column generation, let us consider four results obtained when solving different combinatorial optimization applications on random instances after applying DWD. They are depicted in Figure 3.3. These results consider adding one column per subproblem and using the standard (simplex-based) column generation method.

On the vertical axis, we have the relative gap ($gap = (UB - LB)/(1 + UB)$) while on the horizontal axis the number of iterations is shown. The problems considered are: the vehicle routing problem with time windows (VRPTW) [28, 72, 112], the uncapaciated facility location problem (UFLP) [1, 22, 36]; the cutting stock problem (CSP) [46, 47,



(a) VRPTW-C101

(b) UFLP-B1.1

(c) CSP-NN10

(d) CLSPST-X12119B

Figure 3.3: The standard column generation method behaviour

76]; and the capacitated lot-sizing problem with setup times (CLSPST) [27, 73, 114]. It can be seen that for the instance considered for VRPTW, the heading-in effect is not that pronounced; however, the tailing-off effect is considerable. For the UFLP and CSP instances considered, the behaviour seem similar: a first stage in which the gap is reduced (but rather slowly); then only few iterations are required to obtain a considerable reduction of the relative gap; and finally a very slow convergence close to optimality. In our last example, CLSPST presents a decomposable structure and the subproblems are not identical. This is exploited in our implementation so several columns can be added per iteration and therefore the behaviour seems to be better than for the other applications/instances. However, by looking closer at the results, one can say that: in the first 50 iterations, very little progress was made; in the following 100 iterations, a considerable reduction in the gap was obtained; and finally, it took the standard version around 100 more iterations to converge.

In summary, Figure 3.3 shows general slow progress to optimality at the beginning due to unstable behaviour of the dual variables (heading-in effect), suddenly a (very) steep reduction in the relative gap (the method has gathered enough information to get a good MP representation) and then, slow convergence towards optimality (tailing-off effect).

It is important to emphasize at this point that these two limitations are the two most well-known drawbacks of the standard column generation method and therefore, finding a reliable remedy to them has been a challenge and any progress in this area has always been of great interest to the research community.

3.5 Stabilization strategies

As mentioned in the previous section, column generation is adversely affected by the use of extreme dual solutions. One important observation is that solving every RMP to optimality is not needed in a column generation procedure and, hence, variations of the standard technique avoid it. In general, they rely on interior points of the dual feasible set of the RMP. For instance, the stabilization techniques [17, 32, 88] choose a dual point called the stability centre and add penalization terms to the RMP dual objective function to keep the dual solutions close to this centre and/or add constraints (boxes) so that the dual variables are limited to stay close to a reference point. The modified dual RMP is solved to optimality; however, now the dual solutions do not oscillate so much from one outer iteration to another, because of the penalties and boxes added to the dual problem. These penalties and boxes can be updated at every iteration. The following linear programming model proposed in [32] shows how both stabilization techniques are incorporated in the dual space (see (3.12))

$$\max \quad b^T u - \upsilon_-^T w_- - \upsilon_+^T w_+, \tag{3.16a}$$
$$\text{s.t.} \quad A^T u \leq c, \tag{3.16b}$$
$$u + w_- \geq \delta_-, \tag{3.16c}$$
$$u - w_+ \leq \delta_+, \tag{3.16d}$$
$$w_- \geq 0, \tag{3.16e}$$
$$w_+ \geq 0, \tag{3.16f}$$

where A is the current coefficient matrix of the RMP, w_- and w_+ are additional dual variables and $\delta_-, \delta_+, \upsilon_-$ and υ_+ are vector parameters of suitable dimensions. Note that constraints (3.16c) and (3.16d) restrict the value of u to the interval $[\delta_- - w_-, \delta_+ + w_+]$. Any deviation from the suitable box described by $[\delta_-, \delta_+]$ is captured by w_- and w_+ and penalized by the unitary non-negative penalties υ_- and υ_+. This approach requires: (a) the formulation of the dual problem; (b) the addition of new variables (w_- and w_+); (c) the setting of parameters ($\delta_+, \delta_-, \upsilon_+$ and υ_-); and (d) a strategy to update these parameters.

A different stabilization strategy is proposed in [109]. At each outer iteration, the RMP is solved several times, with different objective coefficients which are randomly generated. Then, a set of optimal solutions in the dual space is obtained and an interior dual point is generated by considering a linear convex combination of these solutions. Rousseau *et al.* present computational results considering instances of VRPTW, for which the number of outer iterations and CPU time were reduced in relation to the standard as well as stabilized column generation. However, for applications with large-scale RMPs, the need to solve these problems several times for different objective functions adversely affects the efficiency of the approach. Moreover, the number of RMPs to be solved per iteration seems to be problem dependent.

A third stabilization technique has been proposed in [120]. Instead of using the optimal dual solutions of the RMP in the subproblems, the *weighted Dantzig-Wolfe method* constructs these dual solutions using a weighted average of the previously constructed dual solutions and the optimal solution of the current RMP. Finiteness of the method has been shown and encouraging computational results are provided for solving the capacitated facility location problem. The dual variables are constructed in the following way

$$u^{k+1} := \frac{1}{w_k} u^k + \frac{w_k - 1}{w_k} \underline{u}^k, \tag{3.17}$$

where u^k is the optimal dual vector in iteration k, and $\underline{u}^k$ is the dual vector associated with the best bound found until iteration k. Additionally, the weight w_k is calculated as

$$w_k := \min\{t, \frac{k+l}{2}\}, \tag{3.18}$$

where t is a threshold such as $t \geq 2$, k is the iteration counter and l is the number of improvements of the lower bound. This dynamic choice of weights allows the method to choose dual variables which are in the neighbourhood of the best dual variables found so far.

Bundle methods [69, 79] have become popular techniques in nondifferentiable optimization and therefore, they have also been studied in the context of column generation/cutting plane methods for solving decomposable mixed-integer programming applications in their relaxed forms [17]. These techniques stabilize the RMP via the use of Euclidean penalties. The following quadratic programming problem demonstrates a modification made to the dual RMP in the bundle method

$$\max \quad b^T u - \frac{1}{2t} \|u - \underline{u}\|^2, \tag{3.19a}$$

$$\text{s.t.} \qquad A^T u \leq c, \tag{3.19b}$$

where t controls the impact in the objective of any deviation from the reference parameter $\underline{u}$ and it is usually updated at every iteration.

A different class of column generation approaches obtains interior points of the dual feasible set without (directly) modifying the RMP. Mitchell and Borchers [96] and Mitchell [94] have addressed the solution of two classes of combinatorial optimization problems by a cutting plane method using the primal-dual interior point method. For those particular applications, the cutting planes are found by full enumeration. Additionally, several heuristic procedures are proposed to improve the solution (primal heuristic), drop cutting planes and fix variables. Notice that these approaches 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) [4, 34, 49, 53] is an interior point approach that relies on central prices. The strategy consists of computing a dual point u which is an approximate analytic centre of the localization set associated with 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 RMP. Note that the ACCPM can cope with second order information. However, since our RMPs have linear objectives, we have simplified its presentation. Let us define the localization set of the dual of the current RMP (3.11) as

$$\mathcal{L}_k = \{u \ : \ b^T u \geq \underline{\theta}, \ A^T u \leq c, \},$$

where A is the coefficient matrix of the current RMP and $\underline{\theta}$ the best bound found so far. Having added slack variables $s_0 \in \mathbb{R}$ and $s_i \in \mathbb{R}^{|\overline{N}|}$ to the two constraints, we define the weighted logarithmic barrier function associated with this localization set as

follows

$$F(s_0, s) = -w_0 \log s_0 - \sum_{i=1}^{|\overline{N}|} w_i \log s_i,$$

where w_0 and w_i for every $i \in \overline{N}$ are non negative barrier penalties. 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 [2, 50, 80]. The use of the ACCPM in a branch-and-bound framework is proposed in [34]. The use of stabilization terms within the analytic centre cutting plane method has been successful, as shown in [5]. The proximal analytic centre cutting plane method solves the following quadratic program

$$\min \quad F(s_0, s) + \frac{\rho}{2}\|u - \underline{u}\|^2, \tag{3.20}$$

$$\text{s.t.} \quad A^T u + s = c, \tag{3.21}$$

$$b^T u - s_0 = \underline{\theta}, \tag{3.22}$$

$$s_0 > 0, \tag{3.23}$$

$$s > 0, \tag{3.24}$$

where ρ is the penalization term for deviating from the best dual variables found so far. The algorithm used to calculate the (proximal) analytic centre is a Newton method. A thorough comparison of the method proposed in this thesis with the implementation of the ACCPM [5] is provided in Chapter 5.

Although other polynomial cutting plane methods are proposed in the literature, no efficient computational implementations are publicly available for them [95].

Another column generation method based on an interior point approach is the primal-dual column generation method (PDCGM). Proposed in [62], the technique uses an infeasible primal-dual interior point method to find non-optimal solutions of the RMPs. In the first column generation iterations, each RMP is solved with a loose tolerance. This tolerance is dynamically adjusted throughout the iterations as the relative gap approaches zero. Gondzio and Sarkissian present promising computational results for the *nonlinear multicommodity network flow* problem, whose linearisation is solved by column generation. A similar strategy is used by [89] to solve linear programming problems by combining Dantzig-Wolfe decomposition and a primal-dual interior point method. Martinson and Tind also report a substantial reduction in the number of outer iterations when compared to other column generation procedures.

To the best of our knowledge, the PDCGM has never been applied in the context of integer programming, where column generation is used to solve linear relaxations that typically arise from Dantzig-Wolfe reformulations.

Chapter 4

A Selection of Problems: Formulation and Decomposition

In this section we describe the problems we have considered in our computational studies. All these problems belong to the (mixed-)integer programming optimization class and their structure can be exploited by the use of the Dantzig-Wolfe decomposition principle. For each application, we state its original formulation, and then describe how the Dantzig-Wolfe decomposition principle is applied to it. We explicitly define the master problem and the subproblems to be used in column generation. For details on the size of the problems, see Appendix A.

Remark For clarity purposes, in the remainder of this chapter, we have adopted a compact representation of vectors. For a given set $I = \{1, \ldots, n\}$ of indices, we denote by $[x_i]_{i \in I}$ the vector $(x_1, \ldots, x_n)$ and the vector $(x_{11}, \ldots, x_{1n}, \ldots, x_{nn})$ is represented by $[x_{ij}]_{i,j \in I}$

4.1 Cutting stock problem

The cutting stock problem (CSP), in its one-dimensional case, can be posed as an integer programming problem which aims to find the smallest number of rolls of width W needed to be cut so demands d_j of pieces of width w_j are satisfied, for every $j \in M = \{1, 2, \ldots, m\}$ [10, 46, 47, 76]. An upper bound n on the number of rolls needed to satisfy the demands is known and an index $i \in N = \{1, 2, \ldots, n\}$ is associated with each roll. A mathematical formulation for CSP is given by [10]

$$\min \quad \sum_{i \in N} y_i, \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ij} \geq d_j, \qquad \forall j \in M, \tag{4.1b}$$

$$\sum_{j \in M} w_j x_{ij} \leq W y_i, \qquad \forall i \in N, \tag{4.1c}$$

$$y_i \in \{0, 1\}, \qquad \forall i \in N, \tag{4.1d}$$

$$x_{ij} \geq 0 \text{ and integer}, \qquad \forall i \in N, \forall j \in M, \tag{4.1e}$$

where $y_i = 1$, if the roll i is used, and 0, otherwise. The integer variable x_{ij} denotes the number of times a piece of width w_j is cut from roll i. Constraints (4.1b) ensure

that all demands are satisfied while constraints (4.1c) guarantee that the sum of the widths of all pieces obtained from a roll does not exceed its width W.

4.1.1 Dantzig-Wolfe decomposition

The coefficient matrix of problem (4.1) has a special structure with coupling constraints given by (4.1b) which links the rolls. Hence, the problem is well-suited to the application of DWD. Consider the set $\mathcal{X}$ as the set which contains all solutions satisfying constraints (4.1c) - (4.1e). Moreover, we define the subset $\mathcal{X}_i$, for each $i \in N$, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_n$ and replace each $\mathcal{X}_i$ by its convex hull $conv(\mathcal{X}_i)$, which is a bounded set and hence can be fully described by convex combinations of its extreme points (see Section (3.1) for details). Let P_i be the set of indices of all extreme points of $conv(\mathcal{X}_i)$ for every $i \in N$. These extreme solutions are then denoted by $(y_p^i, [x_{pj}^i]_{j \in M})$, for each $i \in N$ and $p \in P_i$. Using this notation, we obtain the following master problem

$$\min \quad \sum_{i \in N} \sum_{p \in P_i} y_p^i \lambda_p^i, \tag{4.2a}$$

$$\text{s.t.} \quad \sum_{i \in N} \sum_{p \in P_i} x_{pj}^i \lambda_p^i \geq d_j, \qquad \forall j \in M, \tag{4.2b}$$

$$\sum_{p \in P_i} \lambda_p^i = 1, \qquad \forall i \in N, \tag{4.2c}$$

$$\lambda_p^i \geq 0, \qquad \forall i \in N, \forall p \in P_i. \tag{4.2d}$$

Let us define $u \in \mathbb{R}_+^m$ and $v \in \mathbb{R}^n$ as the dual variables associated with constraints (4.2b) and (4.2c), respectively. The oracle associated with the master problem (4.2) is given by a set of n subproblems. Hence, for every $i \in N$, we have the following subproblem

$$\min \quad y_i - \sum_{j \in M} \overline{u}_j x_{ij} - \overline{v}_i, \tag{4.3a}$$

$$\text{s.t.} \quad (y_i, [x_{ij}]_{j \in M}) \in conv(\mathcal{X}_i), \tag{4.3b}$$

where $\overline{u} = (\overline{u}_1, \ldots, \overline{u}_m)$ and $\overline{v} = (\overline{v}_1, \ldots, \overline{v}_n)$ represent an arbitrary dual solution.

One should note that since the stock pieces are identical (*i.e.*, have the same width W), we have that $conv(\mathcal{X}_1) = conv(\mathcal{X}_2) = \ldots = conv(\mathcal{X}_n)$. This implies that for a fixed dual solution $(\overline{u}, \overline{v})$, the subproblems are the same for every $i \in N$ and that the oracle will generate n identical columns. To avoid this, we aggregate variables in the master problem as explained in Section 3.4.1. The resulting aggregated master problem is given by

$$\min \quad \sum_{p \in P} y_p \lambda_p, \tag{4.4a}$$

$$\text{s.t.} \quad \sum_{p \in P} x_{pj} \lambda_p \geq d_j, \qquad \forall j \in M, \tag{4.4b}$$

$$\lambda_p \geq 0, \qquad \forall p \in P, \tag{4.4c}$$

where set P contains the indices of all extreme solutions of $conv(\bar{\mathcal{X}})$, with $\bar{\mathcal{X}} := \mathcal{X}_1 =$

$\ldots = \mathcal{X}_n$. Also, since $(y_i, [x_{ij}]_{j \in M}) = \mathbf{0}$ is a feasible solution, we have dropped the convexity constraint. This is because n is an inactive upper bound in the constraint $\sum_{p \in P} \lambda_p \leq n$. Moreover, the oracle is now given by only one subproblem, which is the same as (4.3), except for dropping the index i and $\overline{v}_i$. Instead of dealing with a subproblem of form (4.3), it is common that one solves the knapsack problem associated with it, which has the following form

$$\max \quad \sum_{j \in M} \overline{u}_j x_j, \tag{4.5a}$$

$$\text{s.t.} \quad \sum_{j \in M} w_j x_j \leq W, \tag{4.5b}$$

$$x_j \geq 0 \text{ and integer}, \quad \forall j \in M. \tag{4.5c}$$

An optimal solution $(x_1^\star, \ldots, x_m^\star)$ of this subproblem is used to generate a column for (4.4). If $\sum_{j \in M} \overline{u}_j x_j^\star > 1$, then the column is generated by setting $y_p := 1$ and $x_{pj} := x_j^\star$ for all $j \in M$. Otherwise, the column is generated by setting $y_p := 0$ and $x_{pj} := 0$ for all $j \in M$ which represents an empty pattern.

4.2 Vehicle routing problem with time windows

Consider a set of customers $C = \{1, 2, \ldots, n\}$ with demands d_i, for every $i \in C$ which has to be served by a fleet of vehicles represented by set $V = \{1, 2, \ldots, v\}$. Each vehicle has a fixed capacity q and can serve as many customers in a route as long as its capacity is not exceeded by the customers' requirements. Demand d_i, cannot be split and, therefore, each customer $i \in C$ must be served by only one vehicle. Additionally, a vehicle can only service a customer within a time window $[a_i, b_i]$ and usually a service time is considered for every customer. Arrivals after time b_i are not allowed. Also, if a vehicle arrives earlier than a_i, it will need to wait until a_i to serve customer $i \in \mathcal{C}$. We assume all the vehicles are identical (all the same capacity and costs) and are initially at the same depot. Moreover, it is also assumed that every route must finish at the depot. The objective is to design a set of minimum cost routes in order to serve all the customers within their time windows [28, 72, 112].

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, represented by t_{ij}, satisfies the triangle inequality. With all these elements in mind, the vehicle routing problem with time windows (VRPTW) can be formulated as follows

$$\min \quad \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ijk}, \tag{4.6a}$$

$$\text{s.t.} \quad \sum_{k\in V}\sum_{j\in N} x_{ijk} = 1, \qquad \forall i \in C, \tag{4.6b}$$

$$\sum_{i\in C} d_i \sum_{j\in N} x_{ijk} \leq q, \qquad \forall k \in V, \tag{4.6c}$$

$$\sum_{j\in N} x_{0jk} = 1, \qquad \forall k \in V, \tag{4.6d}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0, \qquad \forall h \in C,\ \forall k \in V, \tag{4.6e}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1, \qquad \forall k \in V, \tag{4.6f}$$

$$s_{ik} + w_i + t_{ij} - T_{ij}(1 - x_{ijk}) \leq s_{jk}, \qquad \forall\ i, j \in N,\ \forall k \in V, \tag{4.6g}$$

$$a_i \leq s_{ik} \leq b_i, \qquad \forall i \in N,\ \forall k \in V, \tag{4.6h}$$

$$x_{ijk} \in \{0, 1\}, \qquad \forall\ i, j \in N,\ \forall k \in V. \tag{4.6i}$$

The binary variable x_{ijk} determines whether vehicle $k \in V$ visits customer/depot $i \in N$ and then goes immediately to vertex $j \in N$. The continuous variable s_{ik} represents the time vehicle $k \in V$ starts to serve customer $i \in C$. Usually, the cost c_{ij} is calculated as the Euclidean distance between customers i and j, although other measures can be used. The service time required to serve customer $i \in C$ is denoted by w_i. Constraints (4.6b) together with (4.6i) guarantee that each customer must be visited by only one vehicle. Constraints (4.6c) make sure that a vehicle cannot exceed its capacity. These three sets of constraints together ensure that the demand of each customer is satisfied by only one vehicle. While constraints (4.6d) assure that each vehicle has to start its route at the depot, constraints (4.6f) guarantee that each vehicle has to finish its route at the depot. Constraints (4.6e) ensure 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 (4.6g) link the vehicle departure time from a customer and its immediate successor. Indeed, if $x_{ijk} = 1$ then the constraint becomes $s_{ik}+w_i+t_{ij} \leq s_{jk}$. On the other hand, if $x_{ijk} = 0$ and for a suitable large value of T_{ij}, the constraint is always satisfied. The value of T_{ij} can be defined as $\max\{0, b_i + w_i + t_{ij} - a_j\}$ for every $i, j \in N$. Constraints (4.6h) enforce that the vehicle k serves customer i between time a_i and b_i. Note that if a vehicle is not used, its route is defined as $(0, n + 1)$ with associated cost equal to 0. The parameters considered for this application such as time windows, capacity of the vehicles, demands and costs are non-negative and usually integers.

4.2.1 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 (4.6b) which link all the vehicles. Defining $\mathcal{X}$ as the set of all extreme solutions satisfying constraints (4.6c) - (4.6i) and defining v independent subsets $\mathcal{X}_k$ from $\mathcal{X}$, for each $k \in V$, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_v$, separability is gained. Let us replace each $\mathcal{X}_k$ by its convex hull $conv(\mathcal{X}_k)$ which can be fully represented by its set of extreme points $([x^k_{ijp}]_{i,j\in N}, [s^k_{ip}]_{i\in N})$. For every $k \in V$,

the set of indices of all the extreme solutions of $conv(\mathcal{X}_k)$ is represented by P_k. Then, the resulting master problem is defined as

$$\min \quad \sum_{k\in V}\sum_{i\in N}\sum_{j\in N}\sum_{p\in P_k} c_{ij}x^k_{ijp}\lambda^k_p, \tag{4.7a}$$

$$\text{s.t.} \quad \sum_{k\in V}\sum_{j\in N}\sum_{p\in P_k} x^k_{ijp}\lambda^k_p = 1, \qquad \forall i \in C, \tag{4.7b}$$

$$\sum_{p\in P_k} \lambda^k_p = 1, \qquad \forall k \in V, \tag{4.7c}$$

$$\lambda^k_p \geq 0, \qquad \forall k \in V, \forall p \in P_k, \tag{4.7d}$$

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 q and cost of going from i to j), we have that $conv(\mathcal{X}_1) = conv(\mathcal{X}_2) = \ldots = conv(\mathcal{X}_v)$ and hence the oracle will generate v identical columns. We can avoid this by aggregating variables and using the following master problem

$$\min \quad \sum_{i\in N}\sum_{j\in N}\sum_{p\in P} c_{ij}x_{ijp}\lambda_p, \tag{4.8a}$$

$$\text{s.t.} \quad \sum_{j\in N}\sum_{p\in P} x_{ijp}\lambda_p = 1, \qquad \forall i \in C, \tag{4.8b}$$

$$\lambda_p \geq 0, \qquad \forall p \in P, \tag{4.8c}$$

where P is the set of indices of all extreme points of $conv(\bar{\mathcal{X}})$, with $\bar{\mathcal{X}} := \mathcal{X}_1 = \ldots = \mathcal{X}_v$. Since $([x^k_{ijp}]_{i,j\in N}, [s^k_{ip}]_{i\in N}) = \mathbf{0}$ is a feasible solution the convexity constraint can be dropped since v is a loose upper bound in the constraint $\sum_{p\in P}\lambda_p \leq v$. Let $u \in \mathbb{R}^n$ denote the dual variables associated with constraints (4.8b). 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 (4.8) is given by the subproblem

$$\min \quad \sum_{i\in N}\sum_{j\in N}(c_{ij} - \overline{u}_j)x_{ij}, \tag{4.9a}$$

$$\text{s.t.} \quad ([x_{ij}]_{i,j\in N}, [s_i]_{i\in N}) \in conv(\bar{\mathcal{X}}). \tag{4.9b}$$

Subproblem (4.9) is known as the elementary shortest path problem with resource constraints [72]. An optimal solution $([x^\star_{ij}]_{i,j\in N}, [s^\star_i]_{i\in N})$ of this problem is an extreme point of $conv(\bar{\mathcal{X}})$. To generate a column for (4.8), we set $x_{ijp} = x^\star_{ij}$, for all $i, j \in N$ which turns to be a vector composed of 0's and 1's elements.

4.3 Capacitated lot-sizing problem with setup times

Consider a set of time periods denoted by $N = \{1, \ldots, n\}$ and a set of items denoted by $M = \{1, \ldots, m\}$ 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 [114] this problem can be formulated as

$$\min \quad \sum_{t\in N}\sum_{j\in M}(c_{jt}x_{jt} + h_{jt}s_{jt} + f_{jt}y_{jt}) \tag{4.10a}$$

$$\text{s.t.} \quad \sum_{j\in M}(a_{jt}x_{jt} + b_{jt}y_{jt}) \leq C_t, \qquad \forall t \in N \tag{4.10b}$$

$$s_{j(t-1)} + x_{jt} = d_{jt} + s_{jt}, \qquad \forall j \in M, \forall t \in N, \tag{4.10c}$$

$$x_{jt} \leq Dy_{jt}, \qquad \forall j \in M, \forall t \in N, \tag{4.10d}$$

$$x_{jt} \geq 0, \qquad \forall j \in M,\ \forall t \in N, \tag{4.10e}$$

$$s_{jt} \geq 0, \qquad \forall j \in M,\ \forall t \in N, \tag{4.10f}$$

$$y_{jt} \in \{0,1\}, \qquad \forall j \in M,\ \forall t \in N, \tag{4.10g}$$

where x_{jt} represents the production level of item j in period t and s_{jt} is the number of units in stock of item j at the end of time period t. If $y_{jt} = 1$, item j is produced in time period t, while $y_{jt} = 0$ otherwise. Constraints (4.10b) ensure that for every time period t, the capacity required for the production plan should not exceed the available capacity of the machine in that period. Note that the production plan considers processing and setup times. Constraints (4.10c) are the balance constraints which guarantee that for every item j, the production in period t and units in stock at the beginning of that period are equal to the demand and the units in stock at the end of period t. Note that backlogging is not allowed. Constraints (4.10d) ensure that if item j is produced in period t, then the machine must be set up, where D is a sufficiently large number. Constraints (4.10e) and (4.10f) guarantee that the level of production and stock at each period t for each item j is non-negative. Usually, the initial and final inventory levels for every product j are assumed to be zero (*i.e.*, $s_{j0} = s_{jn} = 0$).

4.3.1 Dantzig-Wolfe decomposition

The coefficient matrix of the above formulation has a special structure. Constraints (4.10b) are the coupling constraints which link the items. Let us define $\mathcal{X}$ as the set of all points satisfying constraints (4.10c) - (4.10g). For each $j \in M$, we can define the subset $\mathcal{X}_j$ such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$. Following the developments discussed in Section 3.1, we replace each $\mathcal{X}_j$ by its convex hull, $conv(\mathcal{X}_j)$, for every $j \in M$ which can be fully represented by its extreme points $[x^j_{pt}, s^j_{pt}, y^j_{pt}]_{t\in N}$ since it is a bounded set. The resulting master problem is

$$\min \quad \sum_{j\in M}\sum_{t\in N}\sum_{p\in P_j}\left(c_{jt}x^j_{pt} + h_{jt}s^j_{pt} + f_{jt}y^j_{pt}\right)\lambda^j_p \tag{4.11a}$$

$$\text{s.t.} \quad \sum_{j\in M}\sum_{p\in P_j}\left(a_{jt}x^j_{pt} + b_{jt}y^j_{pt}\right)\lambda^j_p \leq C_t, \qquad \forall t \in N, \tag{4.11b}$$

$$\sum_{p\in P_j}\lambda^j_p = 1, \qquad \forall j \in M, \tag{4.11c}$$

$$\lambda^j_p \geq 0, \qquad \forall j \in M, \forall p \in P_j, \tag{4.11d}$$

where P_j is the set of indices of all extreme points of $conv(\mathcal{X}_j)$, for every $j \in M$. To obtain the oracle associated with this master problem, let $u \in \mathbb{R}^n_-$ and $v \in \mathbb{R}^m$ denote the dual variables associated with constraints (4.11b) and (4.11c), respectively. Note

that u is restricted to be non-positive. Let $\overline{u} = (\overline{u}_1, \ldots, \overline{u}_n)$ and $\overline{v} = (\overline{v}_1, \ldots, \overline{v}_m)$ be an arbitrary dual solution. The oracle is then defined by m subproblems of the form:

$$\min \quad \sum_{t \in N} ((c_{jt} - a_{jt}\overline{u}_t)x_{jt} + h_{jt}s_{jt} + (f_{jt} - b_{jt}\overline{u}_t)y_{jt}) - \overline{v}_j \tag{4.12a}$$

$$\text{s.t.} \quad [x_{jt}, s_{jt}, y_{jt}]_{t \in N} \in conv(\mathcal{X}_j), \tag{4.12b}$$

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. Moreover, and unlike the other two cases, the subproblems are very likely to be different since the demands and costs are likely to be different. Therefore, m subproblems have to be solved at every iteration and m columns can be added to the RMP at each column generation iteration. 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 (4.11) 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.

Chapter 5

Primal-Dual Column Generation Method

In this chapter we present theoretical and computational developments of the primal-dual column generation method (PDCGM). Firstly, we describe the method and its main components and provide evidence of its convergence. In the second part, we present the results of computational tests whose aim is to compare the performance of the primal-dual column generation method with the standard column generation method (SCGM) and the analytic centre cutting plane method (ACCPM). All the descriptions and results in this chapter closely follow the developments presented in [58].

As discussed briefly in Chapter 3, standard column generation and analytic centre approaches are extremal strategies, as they are based on optimal solutions but for different objective functions. With the former strategy one obtains an optimal vertex solution at every iteration while, with the latter, the solution of the barrier problem associated with the localization set in the dual space. In fact, the analytic centre of a feasible set corresponds to the optimal solution of a modified dual problem associated with the RMP. From this point of view, the primal-dual column generation technique is somewhere in the middle between these two approaches. It relies on solutions that are close to optimality (sub-optimal), but at the same time not far from the central trajectory in the dual feasible set (well-centred).

One of the contributions of using sub-optimal solutions is that fewer inner iterations are usually needed to solve each RMP. This is due to the method not requiring high accuracy at an initial stage. Hence, the running time per outer iteration is usually reduced. Another benefit of this strategy is that the dual variables are not expected to change dramatically from one iteration to another due to the use of well-centred solutions (close to the central path). This provides stability and reduces the heading-in effect. As a consequence, fewer outer iterations as well as less total CPU time will usually be required to solve the MP.

In Figure 5.1, we illustrate the possible dual solutions found by the aforementioned column generation strategies.

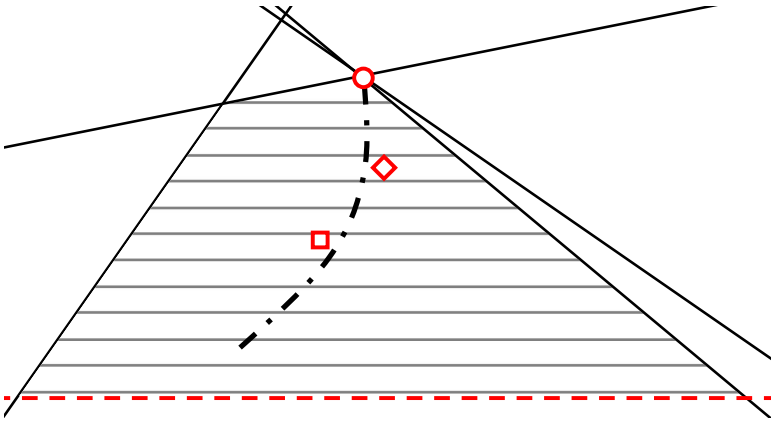


Figure 5.1: Schematic illustration of solutions provided by the SCGM (○), the PDCGM (◇) and the ACCPM (□) strategies in the dual space

The area represented by horizontal lines is the localization set and the dashed line represents the best dual bound found so far. The circle denotes the solution obtained by the standard column generation method based on the simplex method which is at the vertex of the feasible region. On the other hand, the square represents the solution obtained by the analytic centre cutting plane method. The central trajectory (in a very particular space) is denoted by the dashed-dotted curved line while the diamond figure illustrates the solution obtained by the primal-dual column generation method. Note that the solution obtained by the PDCGM becomes closer to the optimal facet of the polytope in the dual space as the method approaches the termination. This is explained in the next section.

The strategy known as the PDCGM was first proposed in [62]. It uses a primal-dual interior point method to get solutions which have a non-zero distance to optimality and at the same time, are well-centred in the feasible set. Note that this is not achievable if one would like to use a simplex-type method since, in general, the distance to optimality is not known in advance.

5.1 Theoretical developments

Recalling the notation introduced in Chapter 3, we consider that a given RMP is represented by (3.11). Its optimal primal-dual solution is denoted by $(\overline{\lambda}, \overline{u})$. As with standard column generation, the PDCGM starts with an initial RMP with enough columns to ensure a bounded solution. Nevertheless, at every iteration, instead of solving the RMP to optimality, a sub-optimal feasible solution $(\tilde{\lambda}, \tilde{u})$ of the current RMP is obtained. This sub-optimal solution is defined as follows.

Definition 5.1. *A primal-dual feasible solution $(\tilde{\lambda}, \tilde{u})$ of the RMP is called a sub-optimal solution, or ε-optimal solution, if it satisfies*

$$0 \leq (c^T\tilde{\lambda} - b^T\tilde{u}) \leq \varepsilon(1 + |c^T\tilde{\lambda}|),$$

for some tolerance $\varepsilon > 0$.

We denote by $\tilde{z}_{RMP} = c^T\tilde{\lambda}$ the objective value corresponding to the sub-optimal solution $(\tilde{\lambda}, \tilde{u})$. Since $c^T\tilde{\lambda} \geq c^T\overline{\lambda} = z_{RMP}$, we have the following inequalities $z_{MP} \leq z_{RMP} \leq \tilde{z}_{RMP}$ and therefore, $\tilde{z}_{RMP}$ is a valid upper bound on the optimal value of the

MP. Once a sub-optimal solution of the RMP is obtained, we send the dual solution $\tilde{u}$ to the oracle. Then the oracle returns either $z_{SP}(\tilde{u}) = 0$, if no columns could be generated using the proposed query point, or $z_{SP}(\tilde{u}) < 0$, together with one or more columns to be added to the RMP. Observe that when $z_{SP}(\tilde{u}) < 0$ at least one column can always be generated, as $\tilde{u}$ is dual feasible for the RMP and, hence, all columns already generated must have a non-negative reduced cost.

Having defined the concept of sub-optimal solutions, we can now show that the bounds provided by using such solutions, are valid in a column generation context.

To do so, let us first consider the value $\kappa > 0$ defined as in (3.14). The value of κ depends on the application; however, it is usually a known parameter. According to the following lemma, a lower bound of the optimal value of the MP can still be obtained. It is the classical Lagrangian bound (see *e.g.* [11, 17]), but derived from a column generation scheme and using a subpotimal solution.

Lemma 5.2. *Let $\tilde{z}_{SP} := z_{SP}(\tilde{u})$ be the value of the oracle corresponding to the sub-optimal 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 (3.10b) 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, we have that $z^\star = c^T\lambda^\star \geq \kappa\tilde{z}_{SP} + b^T\tilde{u}$. □

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

$$\gamma\mu \leq (c_j - u^T a_j)\lambda_j \leq (1/\gamma)\mu, \quad \forall j \in \overline{N}, \tag{5.1}$$

for some $\gamma \in (0, 1)$, where $\mu = (1/|\overline{N}|)(c^T - u^T A)\lambda$. For a given value of γ, condition (5.1) resembles our definition of the symmetric neighbourhood in (2.16), noting that $s_j = c_j - u^T a_j$ and $\lambda_j = x_j$, for every $j \in \overline{N}$.

By imposing (5.1), we ensure that the point is not too close to the boundary of the primal-dual feasible set and, therefore, the oscillation of the dual solutions of two consecutive problems will be relatively small. As pointed out in [57], (5.1) is a natural way of stabilizing the dual solutions if a primal-dual interior point method is used to solve the RMP.

One important observation is that the tolerance ε which controls the distance of $(\tilde{\lambda}, \tilde{u})$ to optimality can be "large" at the beginning of the column generation process, as a very rough approximation of the MP is known at this time. We exploit this fact and during the first iterations, the PDCGM aims to find interesting columns as quickly as possible by solving the RMP to a predefined tolerance. However, solving the RMPs to a loose tolerance is likely to hamper convergence to the optimal MP so at some point along the process this tolerance needs to be tightened to ensure that the method

converges. The best way we have to identify when column generation is converging to an optimal solution of the MP is by using the relative gap in the outer iterations, which is given by

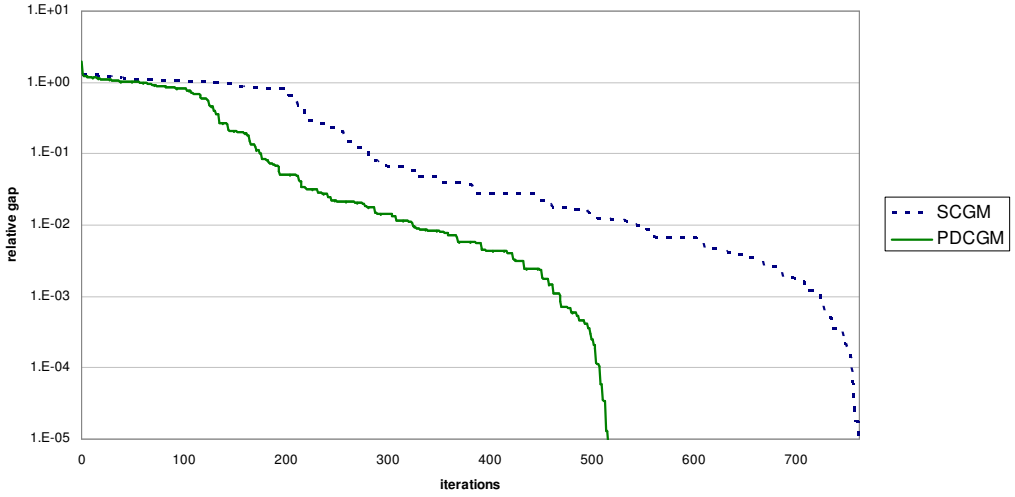
$$gap := \frac{c^T\tilde{\lambda} - (\kappa\tilde{z}_{SP} + b^T\tilde{u})}{1 + |c^T\tilde{\lambda}|}, \tag{5.2}$$

where $\tilde{z}_{SP} := z_{SP}(\tilde{u})$, as defined in Lemma 5.2. Then, at the end of every outer iteration, we recompute the relative gap, and the tolerance ε used in the PDCGM is updated using

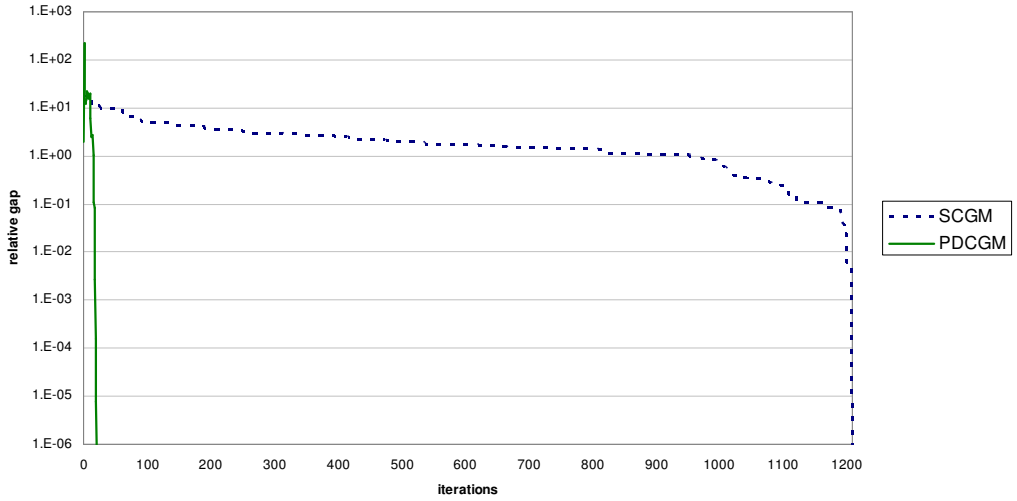
$$\varepsilon^k := \min\{\varepsilon_{\max},\ gap^{k-1}/D\}, \tag{5.3}$$

where $D > 1$ is the *degree of optimality* that links the tolerance ε^k to the relative gap at iteration $k-1$. In our theoretical developments and computational experiments, we consider D as a fixed parameter. Also, a threshold $\varepsilon_{\max}$ is used so that the sub-optimal solution is not too far away from the optimum. Note that the update of the tolerance after reaching the break point is done gradually and we do not require the method to solve every RMP to optimality apart from the last iterations. By using this dynamically adjusted tolerance, we expect to reduce the problems arising when using the standard column generation, namely the heading-in and the tailing-off effects.

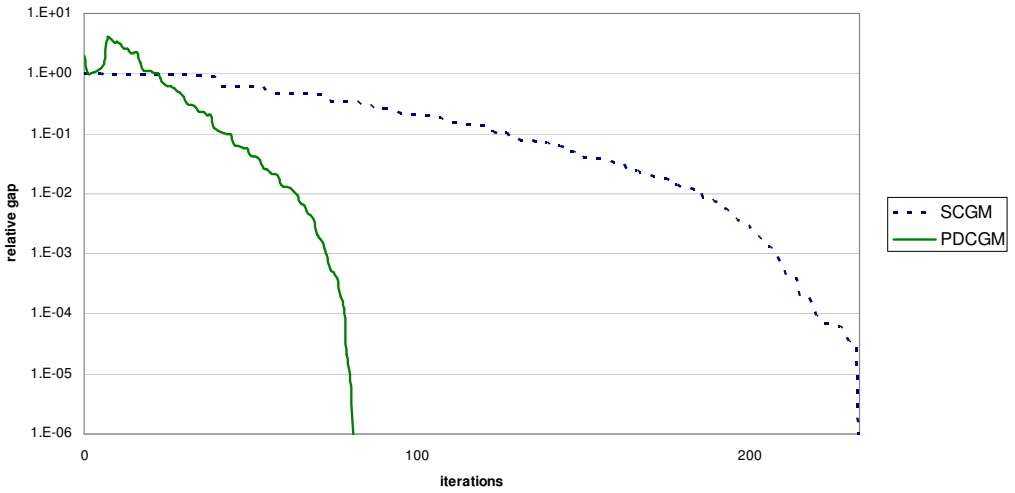
Figure 5.2 shows the behaviour of the PDCGM when solving random instances of the cutting stock problem, the vehicle routing problem with time windows and the capacitated lot-sizing problem with setup times adding one column per iteration. These figures illustrate the changes in the relative duality gap over the iterations in a logarithmic scale. We have included the results obtained with the SCGM. Note that these figures show the results for the same instances used in Figure 3.3. If one considers the heading-in stage, the PDCGM spends fewer iterations than the SCGM. We believe this is due to the use of well-centred dual solutions so columns with richer information are obtained earlier in the process. Once the method has obtained enough information, it converges in fewer iterations than the SCGM reducing the tailing-off effect. One can observe from these figures that the rate of convergence of the PDCGM at this stage is much faster than the rate of convergence of the SCGM. These could be explained again by the use of well-centred dual solutions as well as the use of a primal-dual interior point method and the dynamic adjustment of the tolerance used in the PDCGM. These figures are only a sample of the results obtained with the PDCGM. For more extensive computational experiments, refer to Section 5.2. Note that increments in the relative duality gap with the PDCGM occur due to the use of sub-optimal solutions.



(a) CSP-NN10



(b) VRPTW-C101_100



(c) CLSPST-X12119B

Figure 5.2: Comparison between the standard column generation method and the primal-dual column generation method

During all the process, we pass to subproblems the well-centred solutions provided by the use of the primal-dual interior point method and considering the symmetric neighbourhood.

Notice that unlike the standard approach, $\tilde{z}_{SP} = 0$ does not guarantee convergence of the method since there may still be a difference between $c^T\tilde{\lambda}$ and $b^T\tilde{u}$. Lemma 5.3 below shows that the gap is still reduced in this case, and the progress of the algorithm is guaranteed.

Lemma 5.3. *Let $(\tilde{\lambda}, \tilde{u})$ be the sub-optimal solution of the RMP, found at iteration k with tolerance $\varepsilon^k > 0$. If $\tilde{z}_{SP} = 0$, then the new relative gap is strictly smaller than the previous one, i.e., $gap^k < gap^{k-1}$.*

Proof. We have that $\tilde{z}_{RMP} = c^T\tilde{\lambda}$ is an upper bound of the optimal solution of the MP. Also, from Lemma 5.2 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 from Definition 5.1, the right-hand side of this equality is less or equal than 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 (5.3) $gap^{k-1} \geq D\varepsilon^k > \varepsilon^k$. Otherwise, $\varepsilon^k = gap^{k-1}/D$ and, again, $gap^{k-1} > \varepsilon^k$ which completes the proof. □

Algorithm 5.1 summarizes the main steps of the primal-dual column generation method. Notice that this column generation method has a simple algorithmic description, similar to the standard approach (Algorithm 3.1). Hence, it can be implemented with the same level of difficulty if a primal-dual interior point solver is readily available. Notice that κ is known in advance and is problem dependent. Also, the upper bound of the RMP, $\tilde{z}_{RMP}$, may increase slightly from one iteration to another due to the use of sub-optimal solutions and, hence, we store the best value found so far in UB (Step 3 in Algorithm 5.1).

Algorithm 5.1 The Primal-Dual Column Generation Method

Input: Initial RMP; parameters κ, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
Set: LB $= -\infty$, UB $= \infty$, $gap = \infty$, $\varepsilon = 0.5$;

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

Since the PDCGM relies on sub-optimal solutions of each RMP, it is important to provide guarantees that it is a valid column generation procedure. In other words, we have to ensure that the method is a finite iterative process that delivers an optimal solution, $\lambda^\star$, of the MP. Despite the fact that the optimality tolerance ε decreases geometrically in the algorithm, there is a special case in which the subproblem value,

$z_{SP}(\tilde{u})$, is zero which would cause the PDCGM to stall. However, by using Lemma 5.3 we can ensure that the primal-dual column generation method still converges to the optimal solution of the MP. The proof of convergence is given in Theorem 5.4.

Theorem 5.4. *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*

$$(\tilde{z} - z^\star) < \delta(1 + |\tilde{z}|). \tag{5.4}$$

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

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

2. $\tilde{z}_{SP} = 0$ and no columns have been generated. If additionally we have $\varepsilon^k < \delta$, then from Lemma 5.3 the gap in the current iteration satisfies $gap^k < \delta$, and the algorithm terminates with the sub-optimal solution $(\tilde{\lambda}, \tilde{u})$. Otherwise, we also know from Lemma 5.3 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 $1/D$ which is less than 1 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

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

Since $\kappa \tilde{z}_{SP} + b^T \tilde{u} \leq z^\star$, the inequality (5.4) 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 shown that the PDCGM converges to an optimal solution in a finite number of iterations, we now provide some remarks about its implementation.

Since we require well-centred primal-dual solutions, we rely on a state-of-the-art primal-dual interior point solver that uses a symmetric neighbourhood of the central path. In our implementation, each RMP is solved by the interior point solver HOPDM [54] which keeps the iterates inside a neighbourhood which has the form (5.1). To achieve this, the solver makes use of multiple centrality correctors [21, 55].

Also, an efficient warmstarting technique is crucial for the good performance of a column generation technique based on an interior point method such as the PDCGM. In the course of the column generation process, closely-related problems are solved. Taking advantage of the similarities between these problems could reduce the computational effort of solving a sequence of them. In our implementation, we rely on the reoptimization technique available in the solver HOPDM [56]. The main idea of this strategy consists of storing a near-optimal and well-centred iterate when solving a given

RMP. After a modification is carried out on the RMP (addition of new columns), the stored point is adjusted to create a full-dimensional initial point to start from. Warm-starting is a key aspect of a successful column generation implementation. Therefore, in Chapter 6 we include a thorough analysis of this feature and propose a new warm-starting strategy in this context.

Finally, obtaining sub-optimal primal-dual solutions is a key feature of the PDCGM. Hence a primal-dual interior point method is well-suited for the implementation of the PDCGM. Moreover, standard simplex-based methods cannot straightforwardly provide sub-optimal solutions nor 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, simplex-based approaches do not provide control on the infeasibilities of the solutions before optimality is reached.

Before moving to the next section of the chapter, it is important to clarify that the PDCGM is more than just a strategy which replaces the simplex method with a primal-dual interior point method in column generation. It has to be understood as a new column generation strategy that exploits the rich information provided by a primal-dual interior point method (distance to optimality) to obtain sub-optimal solutions as needed. Also, the dynamic adjustment of the tolerance ensures that the method does not stall and converges to the optimal solution. Finally, the method requires only two parameters to be set which have straightforward meanings: ε_{max} defines the tolerance for the initial iterations when a loose approximation of the MP is at hand, while D, determines how fast we would like to approach optimality.

5.2 Computational study

In this section we present extensive numerical results obtained by using different column generation strategies for different applications. Note that the results included here follow a different presentation than the ones discussed in [58].

As benchmarks we have chosen three different applications which are well known in the column generation literature and have been described already in Chapter 4: the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST). A description of the instances used in this study and the class they belong to is available in Appendix A. For each application, we have implemented three different column generation strategies which were already described. A brief summary of each strategy and some implementation considerations are given below:

- Standard column generation method (SCGM): each RMP is solved to optimality by a simplex-type method to obtain an extreme optimal dual solution. The implementation closely follows the steps presented in Algorithm 3.1. We have used one of the best available linear programming solvers, IBM ILOG CPLEX v.12.1 [71] to obtain such a solution. Preliminary tests using the default settings for each solver show that the primal simplex method is 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 (with crossover) was inferior to the other two methods.

- Primal-dual column generation method (PDCGM): the sub-optimal solutions of each RMP are obtained by using the interior point solver HOPDM [54], which

is able to efficiently provide well-centred sub-optimal dual points. The algorithm has been already described in Algorithm 5.1.

- Analytic centre cutting plane method (ACCPM): the dual point at each iteration is an approximate analytic centre of the localization set associated with the current RMP described in Section 3.5. The applications were implemented on top of the open-source solver OBOE/COIN [19], a state-of-the-art implementation of the analytic centre strategy with additional stabilization terms [5].

For each application and for the aforementioned column generation strategies, the subproblems are solved using the same source code. We provide more details of the oracle solvers used in each application later. Also, the SCGM 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 better performance of the method on average. We have used different initial dual points for each application, as specified later. All the computational experiments in this section have been obtained using a computer with a 2.26 Ghz Intel Core 2 Duo processor, 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 with the SCGM is to provide evidence of how much can be gained in overall performance in relation to the standard approach which uses extreme dual solutions without any stabilization. It is important to note that due to degeneracy issues an extreme optimal dual solution obtained by the SCGM can be at any vertex of the optimal facet of the feasible polyhedra. Undoubtedly, it would be interesting to consider stabilized versions of standard column generation in our computational comparisons. However, the lack of publicly available codes of stabilized versions discouraged us from including them. For the interested reader, available comparisons between standard and stabilized column generation are available in the literature for the same applications [11, 17, 109]. A brief discussion about some of these results is presented in Section 5.2.5.

Additionally, we have included the ACCPM in our experiments for being a strategy that also relies on an interior point method (although essentially different) providing well-centred dual solutions.

Before continuing, the performance of interior point methods in column generation by solving each RMP to optimality by a state-of-the-art solver (HOPDM) has been tested. The results obtained were not better than the ones obtained by the PDCGM which shows that an appropriate use of an interior point method is essential for its success in the column generation context.

In the applications addressed in this thesis, the column generation schemes are obtained by applying the Dantzig-Wolfe decomposition principle to the corresponding integer programming formulations [25, 116]. In each application, the decomposition leads to an integer MP and also an integer (pricing) subproblem. As shown in Chapter 4, we relax the integrality of the variables in the integer MP and then solve it by column generation, which gives a lower bound on the optimal value of the original formulation. To obtain an integer solution, it would be necessary to combine column generation with a branch-and-bound search, which is called a branch-and-price method [8, 86]. However, this combination is beyond the scope of this thesis, as we are concerned with the behaviour of the column generation strategies. A very recent attempt of combining

the primal-dual column generation method in a branch-and-price context to solve the vehicle routing problem with time windows can be found in [98]. This study shows encouraging results when compared to the state-of-the-art method to solve this class of problems.

5.2.1 Cutting stock problem

To analyse the performance of the different column generation strategies addressed here, we have initially selected 262 instances from the one-dimensional CSP literature. All the instances were obtained from `http://www.math.tu-dresden.de/~capad/`.

For this application, 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 [82], the implementation of which was provided by the Leão.

Adding one column to the RMP per iteration In the first set of numerical experiments we consider that only one column is generated by the subproblem solver at each iteration. We have classified the instances into different classes. Table 5.1 presents for each class and strategy: the average number of outer iterations (ite), the average CPU time spent in the oracle in seconds (or(s)) and the average CPU time required for the column generation procedure in seconds (tot(s)). The number of instances per class is shown in column #. Additionally, the last four columns show the ratio between the extreme strategies and the PDCGM.

From the results of our first set of experiments, it seems clear that the PDCGM does not offer any savings in terms of CPU time compared to the SCGM when "easy" instances are solved (classes MTP0xJES, MTP0, hard28, 7hard, 53NIRUPs and gau3). Note that these classes are considered "easy" since the total time required to solve the subproblems (oracle time) for all the strategies is less than 1 second on average. Among these classes, the SCGM is the strategy that offers the best performance if total CPU time is considered. Note that the value of m (number of different widths) included in this class varies from 20 to 189. For all these classes, the PDCGM is the strategy that achieves the smallest number of iterations on average. Similar results are obtained when considering the class with most instances (mX), where m varies from 197 to 200. For this class, the oracle solver requires more time to find the columns and, therefore, the savings in terms of iterations start to pay off. Finally, if we consider class U, with m varying from 15 to 585, the best overall performance is provided by the PDCGM in both performance measures, number of iteration and total CPU time. Note that for this class the three strategies require a considerable time to solve the subproblems and, therefore, savings in number of iterations have an important impact on the total CPU time. The ACCPM does not offer any benefit in any class when compared to the other two strategies.

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 SCGM 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 implementation available in CPLEX 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.

Table 5.1: Average results on 262 instances of CSP for the SCGM, the PDCGM and the ACCPM adding one column at a time.

		SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
class	**#**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
U	20	975.6	632.1	636.8	782.7	134.4	154.7	871.5	582.9	694.7	1.2	4.1	1.1	4.5
mX	145	803.6	4.1	5.2	504.2	4.2	9.2	651.6	6.3	20.5	1.6	0.6	1.3	2.2
MTP0xJES	3	379.7	0.8	0.9	244.7	0.8	1.6	294.0	0.8	2.2	1.6	0.6	1.2	1.4
MTP0	5	383.0	0.7	0.8	264.0	0.6	1.5	303.6	0.7	2.4	1.5	0.5	1.2	1.6
hard28	28	535.7	0.3	0.8	386.4	0.5	2.9	475.8	0.7	6.1	1.4	0.3	1.2	2.1
7hard	7	390.6	0.2	0.4	275.1	0.2	1.4	342.9	0.3	2.6	1.4	0.3	1.2	1.8
53NIRUPs	53	356.6	0.2	0.4	221.0	0.2	1.1	273.5	0.3	2.1	1.6	0.3	1.2	1.8
gau3	1	87.0	0.0	0.0	73.0	0.0	0.2	116.0	0.0	0.1	1.2	0.1	1.6	0.7

Adding k-best columns to the RMP per iteration The knapsack solver used to solve CSP subproblems 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 for CSP in which we have tested these strategies for three different values of k.

Table 5.2: Average results on 262 instances of CSP for the SCGM, the PDCGM and the ACCPM adding up to k columns at a time.

		SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
k	**class**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
	U	304.4	311.9	315.0	176.2	57.4	62.5	293.4	328.8	435.9	1.7	5.0	1.7	7.0
	mX	219.6	1.9	2.5	143.0	1.7	4.0	365.6	5.3	53.0	1.5	0.6	2.6	13.1
	MTP0xJES	82.0	0.3	0.3	60.0	0.3	0.6	133.3	0.7	2.3	1.4	0.6	2.2	3.9
	MTP0	84.2	0.2	0.3	61.8	0.3	0.6	137.6	0.6	2.9	1.4	0.5	2.2	4.8
10	hard28	149.1	0.2	0.5	110.7	0.3	1.4	257.9	0.8	16.1	1.3	0.4	2.3	11.7
	7hard	85.3	0.1	0.2	70.0	0.1	0.5	170.6	0.3	4.7	1.2	0.4	2.4	8.7
	53NIRUPs	82.6	0.1	0.2	59.2	0.1	0.5	141.9	0.3	4.1	1.4	0.4	2.4	8.5
	gau3	25.0	0.0	0.0	21.0	0.0	0.1	78.0	0.0	0.3	1.2	0.3	3.7	4.3
	U	186.8	223.8	228.2	103.2	64.0	69.7	277.6	106.4	372.5	1.8	3.3	2.7	5.3
	mX	109.8	3.9	4.5	89.1	4.2	7.6	408.4	23.4	223.2	1.2	0.6	4.6	29.2
	MTP0xJES	35.3	0.3	0.4	35.0	0.6	0.8	122.3	1.9	4.8	1.0	0.5	3.5	5.7
	MTP0	31.8	0.3	0.3	34.6	0.5	0.8	137.8	1.7	7.1	0.9	0.4	4.0	9.2
50	hard28	66.5	0.6	0.9	68.8	0.9	2.4	279.0	4.4	57.3	1.0	0.4	4.1	23.9
	7hard	37.0	0.2	0.3	42.4	0.4	0.9	183.1	1.6	14.6	0.9	0.4	4.3	16.6
	53NIRUPs	33.8	0.2	0.3	35.8	0.3	0.7	148.8	1.3	12.6	0.9	0.4	4.2	17.4
	gau3	13.0	0.0	0.0	20.0	0.0	0.1	84.0	0.2	0.7	0.7	0.2	4.2	5.7
	U	137.6	248.5	252.5	85.2	72.1	79.3	291.6	158.1	548.3	1.6	3.2	3.4	6.9
	mX	84.8	8.4	9.0	76.4	9.6	14.9	464.2	68.1	468.8	1.1	0.6	6.1	31.5
	MTP0xJES	22.7	0.4	0.5	28.7	1.0	1.3	131.3	3.6	8.3	0.8	0.4	4.6	6.6
	MTP0	25.4	0.5	0.5	28.4	0.8	1.1	144.0	3.7	11.0	0.9	0.5	5.1	9.7
100	hard28	48.2	1.3	1.5	57.1	2.2	4.2	304.1	13.3	105.5	0.8	0.4	5.3	25.2
	7hard	27.0	0.5	0.6	37.4	0.9	1.5	198.3	4.3	35.7	0.7	0.4	5.3	23.6
	53NIRUPs	23.5	0.4	0.5	30.5	0.7	1.2	158.4	3.4	25.7	0.8	0.4	5.2	21.4
	gau3	8.0	0.0	0.0	17.0	0.1	0.2	87.0	0.3	3.6	0.5	0.2	5.1	21.1

In Table 5.2, we present the results obtained by adding more than one column at

each iteration. For the "easy" classes, the SCGM is more efficient than the PDCGM and the ACCPM, regardless the number of columns added at each iteration. Similar results are obtained when class mX is considered. However, when instances in class U are solved, the PDCGM is on average more efficient than the SCGM and the ACCPM in terms of both outer iterations and CPU time. For instance, if we consider $k = 100$, the PDCGM is 3.2 times faster than the SCGM and 6.9 times faster than the ACCPM. 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. A discussion about the warmstarting strategy proposed for the ACCPM is included in Chapter 6.

To conclude this initial set of experiments, it has been observed that the PDCGM outperforms the other two strategies in solving the instances which challenge the column generation strategies (class U). In order to study whether this relative performance can be extended to even larger and more difficult instances, we have performed a second set of experiments.

Additionally, and in order to study the impact of the size of the problems on the different strategy behaviours we have further selected 14 large instances from `http://www.math.tu-dresden.de/~capad/` and compared the performance of the three column generation approaches. These instances have m varying from 615 to 1005, which leads to larger restricted master problems and also larger subproblems. Table 5.3 shows the results of this experiment when 100 columns are added per iteration. In all cases, the PDCGM is faster and requires fewer iterations than the SCGM and the ACCPM, which supports the conclusion that the relative performance of the PDCGM is improved as the instances become larger and more difficult.

Table 5.3: Results on 14 large instances of CSP for the SCGM, the PDCGM and the ACCPM adding up to 100 columns at a time.

		SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
name	**m**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
U09498	1005	548	12760	12947	293	5545	5678	762	10054	21254	1.9	2.3	2.6	3.7
U09513	975	518	9741	9904	267	4169	4277	779	7404	19362	1.9	2.3	2.9	4.5
U09528	945	541	9011	9173	276	4811	4924	740	6586	15920	2.0	1.9	2.7	3.2
U09543	915	506	7676	7798	263	3624	3724	723	5255	13449	1.9	2.1	2.7	3.6
U09558	885	482	5479	5585	265	2631	2730	683	4222	10861	1.8	2.0	2.6	4.0
U09573	855	473	4694	4771	230	1980	2054	672	3732	9794	2.1	2.3	2.9	4.8
U09588	825	467	4876	4950	247	1574	1649	658	3983	9376	1.9	3.0	2.7	5.7
U09603	795	465	3894	3962	237	1598	1668	627	3055	7504	2.0	2.4	2.6	4.5
U09618	765	424	2773	2830	203	1042	1092	617	2156	6467	2.1	2.6	3.0	5.9
U09633	735	432	2833	2878	217	912	969	595	1751	5308	2.0	3.0	2.7	5.5
U09648	705	424	2611	2659	209	808	857	582	1403	4466	2.0	3.1	2.8	5.2
U09663	675	381	2156	2187	202	613	654	534	1074	3325	1.9	3.3	2.6	5.1
U09678	645	376	1745	1775	173	387	418	542	1043	3395	2.2	4.3	3.1	8.1
U09693	615	384	1324	1347	165	401	427	520	876	2773	2.3	3.2	3.2	6.5

5.2.2 Vehicle routing problem with time windows

In order to test the behaviour of the different column generation strategies for VRPTW, we have selected 87 instances from the literature (`http://www2.imm.dtu.dk/~jla/solomon.html`), which were originally proposed in [112]. 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 application. Note that although several algorithms are available in the literature for solving the pricing problem (see [72] for a survey), solving it to optimality may require a relatively large CPU time, especially when the time windows are wide. As a consequence, a relaxed version is solved in practice, in which non-elementary paths are allowed (*i.e.*, paths that visit the same customer more than 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. In all our implementations, the subproblem is solved by our own implementation of the bounded bidirectional dynamic programming algorithm proposed in [107], with state-space relaxation and identification of unreachable nodes [37]. For more details about this implementation, we refer the reader to [98].

Adding one column to the RMP In Table 5.4 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 (ite), the average CPU time to solve the subproblems in seconds (or(s)) and the average total CPU time required for the column generation method in seconds (tot(s)). Column # contains the number of instances per class. In the last four columns, the ratios between the extreme strategies and the PDCGM in terms of outer iterations and total CPU time, are presented. The instances are grouped in terms of the distribution of the customers (C: cluster; R: random; RC: a combination of both) and number of customers (25, 50 and 100). For instance, class C50 contains instances in which 50 customers are distributed in clusters.

For all the classes, the PDCGM shows the best average performance in every class 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 SCGM and the other two strategies increases as well, with the SCGM being the one which shows the worst overall performance. Considering the most challenging instances (*i.e.,* C100, R100 and RC100), the PDCGM is on average between 3.0 and 6.4 and between 1.0 and 1.4 times faster than the SCGM and the ACCPM, respectively.

Notice that, differently from what was observed on CSP results, the CPU time required for solving the RMPs (difference between tot(s) and or(s)) is very small not only for the SCGM, but also for the PDCGM and the ACCPM. For VRPTW, the RMPs have the set covering structure, which corresponds to a very sparse coefficient matrix, a property that is well exploited by the solvers.

Table 5.4: Average results on 87 instances of VRPTW for the SCGM, the PDCGM and the ACCPM adding one column at a time.

		SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
class	#	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
C25	9	142.3	1.0	1.0	35.7	0.3	0.3	53.6	0.3	0.4	4.0	3.1	1.5	1.1
R25	12	77.8	0.3	0.4	52.9	0.2	0.3	146.0	0.2	0.4	1.5	1.1	2.8	1.1
RC25	8	85.3	1.1	1.2	57.1	0.8	0.9	107.0	1.0	1.2	1.5	1.3	1.9	1.4
C50	9	446.7	32.1	32.3	60.3	4.4	4.5	74.6	4.4	4.6	7.4	7.1	1.2	1.0
R50	12	211.6	12.2	12.3	122.7	4.8	5.1	214.9	6.4	6.8	1.7	2.4	1.8	1.3
RC50	8	193.8	18.2	18.3	115.5	9.8	10.1	182.6	12.7	13.1	1.7	1.8	1.6	1.3
C100	9	1049.7	333.8	334.8	115.8	51.9	52.3	127.9	50.0	50.4	9.1	6.4	1.1	1.0
R100	12	700.6	549.0	549.7	260.2	157.5	158.4	375.6	205.9	207.8	2.7	3.5	1.4	1.3
RC100	8	660.1	503.5	504.1	254.1	168.7	169.5	351.9	227.4	229.2	2.6	3.0	1.4	1.4

Adding k-best columns to the RMP Since the subproblem solver is able to provide the k-best solutions at each iteration, we have run a second set of experiments. For each column generation method, we have solved each instance with k equal to 10, 50, 100, 200 and 300. In Table 5.5 we show the results of these experiments where column k denotes the maximum number of columns added at each iteration to the RMP.

For the three classes with 25 customers (C25, R25 and RC25), the SCGM and the PDCGM have a similar overall performance up to a point in which the SCGM outperforms the PDCGM. This is due to the fact that the RMPs are solved more efficiently by the solver in CPLEX than HOPDM. However, it is important to note that all the instances in these classes, are solved in less than 1 second on average by the two strategies. Now, if we take into account classes with 50 and 100 customers, the results obtained considering the total CPU time show that the PDCGM is consistently more efficient than the SCGM for every k. In terms of column generation iterations, the results obtained with the PDCGM and the SCGM for different values of k when 50 customers are considered are similar and there is not a clear winner. However, if larger instances are considered (*i.e.,* 100 customers), the PDCGM outperforms the SCGM. On the other hand, the PDCGM is consistently better than the ACCPM in both performance measures, for all the classes and for all the choices of k. Note that the performance of the ACCPM seems to be unaffected by the number of columns added per iteration.

Table 5.5: Average results on 87 instances of VRPTW for the SCGM, the PDCGM and the ACCPM adding up to k columns at a time.

		SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
k	**class**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
10	C25	35.8	0.4	0.4	18.9	0.1	0.2	44.3	0.2	0.3	1.9	2.2	2.3	1.6
	R25	19.3	0.1	0.1	22.9	0.1	0.2	126.9	0.1	0.3	0.8	0.7	5.5	1.8
	RC25	25.6	0.4	0.4	25.1	0.3	0.4	99.1	0.9	1.0	1.0	1.1	3.9	2.7
	C50	101.1	9.4	9.5	28.1	1.8	1.9	56.8	3.3	3.4	3.6	4.9	2.0	1.8
	R50	49.8	3.9	3.9	40.1	2.0	2.2	158.0	4.4	4.8	1.2	1.8	3.9	2.2
	RC50	53.4	5.8	5.9	45.0	3.8	3.9	141.3	9.3	9.6	1.2	1.5	3.1	2.4
	C100	271.9	93.5	94.3	49.3	16.5	16.9	91.7	32.0	32.5	5.5	5.6	1.9	1.9
	R100	152.4	125.4	125.8	86.2	52.0	52.7	205.5	105.2	106.9	1.8	2.4	2.4	2.0
	RC100	147.8	118.3	118.7	78.3	52.0	52.6	207.1	135.7	137.1	1.9	2.3	2.6	2.6
50	C25	20.0	0.3	0.3	17.8	0.1	0.2	43.1	0.2	0.3	1.1	1.4	2.4	1.5
	R25	10.2	0.1	0.1	17.3	0.1	0.1	126.8	0.1	0.3	0.6	0.6	7.4	2.2
	RC25	14.5	0.3	0.3	19.6	0.2	0.3	95.4	0.9	1.1	0.7	0.8	4.9	3.3
	C50	49.6	5.2	5.3	23.0	1.4	1.6	55.1	3.1	3.3	2.2	3.3	2.4	2.1
	R50	24.3	2.4	2.4	26.7	1.3	1.5	155.8	4.3	4.7	0.9	1.5	5.8	3.0
	RC50	27.8	3.2	3.2	29.8	2.3	2.5	139.3	9.4	9.8	0.9	1.3	4.7	3.8
	C100	131.0	46.8	47.7	40.6	12.9	13.6	88.4	31.0	32.0	3.2	3.5	2.2	2.4
	R100	68.2	58.8	59.3	55.3	32.3	33.6	195.9	95.1	97.9	1.2	1.8	3.5	2.9
	RC100	69.5	58.0	58.4	47.5	31.4	32.2	205.4	133.6	135.4	1.5	1.8	4.3	4.2
100	C25	16.6	0.2	0.2	16.6	0.1	0.2	43.9	0.2	0.4	1.0	1.2	2.7	1.8
	R25	8.8	0.1	0.1	15.8	0.1	0.2	126.8	0.1	0.3	0.6	0.5	8.1	2.1
	RC25	12.3	0.2	0.2	18.4	0.2	0.3	95.1	0.9	1.1	0.7	0.7	5.2	3.3
	C50	39.3	4.4	4.5	21.0	1.5	1.8	55.6	3.2	3.6	1.9	2.4	2.6	2.0
	R50	18.5	2.0	2.0	23.4	1.1	1.4	154.5	4.2	4.8	0.8	1.4	6.6	3.5
	RC50	22.3	2.8	2.8	25.3	1.8	2.1	139.5	9.5	10.0	0.9	1.4	5.5	4.9
	C100	94.7	35.2	36.3	31.8	11.8	12.8	90.9	31.1	32.9	3.0	2.8	2.9	2.6
	R100	53.1	45.7	46.2	40.0	23.3	24.6	196.8	94.2	98.2	1.3	1.9	4.9	4.0
	RC100	50.9	43.0	43.4	41.6	25.3	26.5	204.4	130.2	132.7	1.2	1.6	4.9	5.0
200	C25	13.6	0.2	0.2	15.9	0.1	0.3	44.7	0.2	0.4	0.9	0.8	2.8	1.6
	R25	7.0	0.1	0.1	15.1	0.1	0.2	126.8	0.2	0.3	0.5	0.4	8.4	1.8
	RC25	9.9	0.2	0.2	18.0	0.2	0.4	94.4	0.9	1.1	0.5	0.5	5.2	2.8
	C50	31.1	3.5	3.6	19.3	1.1	1.6	57.6	3.0	3.8	1.6	2.3	3.0	2.4
	R50	14.8	1.6	1.7	21.5	1.0	1.4	155.6	4.2	4.9	0.7	1.2	7.2	3.5
	RC50	18.1	2.1	2.1	22.6	1.6	2.1	140.0	9.6	10.2	0.8	1.0	6.2	4.9
	C100	69.3	27.0	28.2	27.9	9.7	11.2	92.7	31.0	34.9	2.5	2.5	3.3	3.1
	R100	41.4	36.3	36.9	35.0	19.5	21.5	198.8	90.3	100.8	1.2	1.7	5.7	4.7
	RC100	41.6	35.4	35.9	33.5	21.1	22.7	204.3	127.5	132.8	1.2	1.6	6.1	5.8
300	C25	12.3	0.2	0.2	16.3	0.1	0.4	46.1	0.2	0.5	0.8	0.5	2.8	1.3
	R25	6.6	0.1	0.1	14.3	0.1	0.2	126.8	0.1	0.3	0.5	0.3	8.8	1.6
	RC25	10.3	0.2	0.2	17.4	0.2	0.5	96.0	0.9	1.2	0.6	0.4	5.5	2.5
	C50	26.3	3.0	3.2	19.1	1.2	2.0	58.6	2.9	4.1	1.4	1.6	3.1	2.0
	R50	12.8	1.5	1.5	20.6	0.9	1.4	155.5	4.2	5.1	0.6	1.1	7.6	3.6
	RC50	16.4	2.1	2.1	21.8	1.6	2.2	140.0	9.2	9.9	0.8	1.0	6.4	4.5
	C100	57.0	24.2	25.8	28.0	9.5	11.9	96.6	31.2	38.6	2.0	2.2	3.4	3.2
	R100	35.7	30.8	31.5	33.4	18.7	21.7	197.6	85.4	100.3	1.1	1.5	5.9	4.6
	RC100	36.9	30.7	31.3	32.6	20.0	22.3	206.4	124.6	132.0	1.1	1.4	6.3	5.9

Additionally, we have tested the three described column generation strategies in more challenging instances with 200, 400 and 600 customers, which were proposed in [70]. Table 5.6 shows the results of this third round of experiments, adding 300 columns per iteration. The columns have the same meaning as in Table 5.4. For all these instances, the PDCGM requires less CPU time and fewer iterations when compared with the SCGM and the ACCPM. For the most difficult instance the PDCGM is 2.1 and 6.4 times faster than the SCGM and the ACCPM, respectively.

Finally, the benefits of using the PDCGM for solving the relaxation of VRPTW after applying DWD are accentuated with the size and difficulty of the instances, so

Table 5.6: Results on 9 large instances of VRPTW for the SCGM, the PDCGM and the ACCPM adding 300 columns at a time.

	SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
name	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
C200	85	33	41	29	13	15	169	72	82	2.9	2.7	5.8	5.4
R200	57	36	43	45	26	34	423	192	202	1.3	1.2	9.4	5.9
RC200	67	105	110	57	77	88	385	567	607	1.2	1.2	6.8	6.9
C400	137	453	552	53	171	186	272	886	909	2.6	3.0	5.1	4.9
R400	131	793	865	84	596	640	636	2994	3076	1.6	1.4	7.6	4.8
RC400	189	2706	2789	113	1360	1436	521	6548	6649	1.7	1.9	4.6	4.6
C600	183	1921	2335	48	496	510	482	5115	5173	3.8	4.6	10.0	10.1
R600	222	7226	7558	118	4142	4260	897	25599	25870	1.9	1.8	7.6	6.1
RC600	258	18701	18972	150	8677	8844	923	56177	56683	1.7	2.1	6.2	6.4

the larger the instance, the larger the benefits of using this column generation strategy.

5.2.3 Capacitated lot-sizing problem with setup times

In order to cover a wider spectrum of applications, we have considered the capacitated lot-sizing problem with setup times described in Chapter 4. The problem decomposes naturally in blocks for different items and therefore m different subproblems are obtained (one per item). This allows testing of column generation strategies when m essentially different columns are added in a disaggregated framework which differs from the previous two applications. 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 [119].

We have selected 751 instances proposed in [114] to test the aforementioned column generation strategies. The SCGM 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 1 in the convexity constraints. In the ACCPM approach, after several settings, we have chosen $u^0 = 10.0e$ as the initial dual point. For all the column generation strategies we use the same subproblem solver which is our own implementation of the Wagner-Whitin algorithm [119].

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 5.7, we show a summary of our findings using these modified instances. We have grouped the instances in seven different classes. Small instances are included in classes E, F and W while classes G, X1, X2 and X3 contain larger instances. Instances in classes E and F contain 6 items and 75 time periods while instances in class W have 4 or 8 items and 75 time periods. In class G, the instances have 6, 12 or 24 items and 75 or 150 time periods. Classes X1, X2 and X3 contain instances with 100 time periods and 10, 20 and 30 items, respectively.

For each class and strategy we present: the number of column generation iterations (ite), the average CPU time required to solve the subproblems in seconds (or(s)) and

Table 5.7: Average results on 751 instances of CLSPST for the SCGM, the PDCGM and the ACCPM adding one column per subproblem at a time.

		SCGM			PDCGM			ACCPM[1]			SCGM/PDCGM		ACCPM/PDCGM	
class	**#**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
E	58	38.1	0.7	0.7	29.7	0.5	0.9	38.3	0.7	0.8	1.3	0.9	1.3	1.0
F	70	33.4	0.6	0.6	28.0	0.5	0.8	40.4	0.7	0.9	1.2	0.8	1.4	1.1
W	12	66.4	1.2	1.2	55.3	1.0	1.8	48.6	0.8	1.1	1.2	0.7	0.9	0.6
G	71	44.8	6.6	6.6	32.4	3.9	4.7	43.2	5.2	5.6	1.4	1.4	1.3	1.2
X1	180	47.5	4.2	4.2	28.8	2.4	3.0	35.2	3.0	3.3	1.7	1.4	1.2	1.1
X2	180	42.6	7.4	7.5	20.5	3.5	3.9	27.4	4.6	5.0	2.1	1.9	1.3	1.3
X3	180	48.9	12.7	12.8	18.7	4.7	5.2	24.3	6.1	6.7	2.6	2.5	1.3	1.3

[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 $\delta = 10^{-5}$

the average total CPU time required for the column generation procedure (tot(s)) in seconds. Column # indicates the number of instances per class.

From Table 5.7, we can conclude that the strategies have different performances for the classes with small instances 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 SCGM is slightly better for classes E and F, and the ACCPM outperforms the other two strategies only in class W. Considering the oracle times, we observe that for small instances the PDCGM outperforms the SCGM due to the reduction in the number of outer iterations. However this reduction is somehow lost due to the fact that the PDCGM requires considerable time to solve the RMPs while the time required by the SCGM is negligible. Now, if we observe the performance of the strategies on the classes with larger instances (*i.e.*, G, X1, X2 and X3), the PDCGM outperforms the other two strategies on average in both performance measures.

In addition to the previous experiment, we have created a set of more challenging instances. We have taken 3 instances from [114], which were used in [27] as a comparison set. Additionally, we have selected 8 instances from the sets of larger classes, X2 and X3. This small set of 11 instances (*i.e.,* G30, G53, G57, X21117A, X21117B, X21118A, X21118B, X31117A, X31117B, X31118A, X31118B) has been replicated 5, 10, 15 and 20 times following the same procedure described above. The summary of our findings is presented in Table 5.8, 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 SCGM. The PDCGM, and depending on the class, is between 2.7 and 3.0 and between 2.1 and 2.6 times faster than the SCGM and the ACCPM, respectively.

Table 5.8: Average results on 11 modified instances of CLSPST for the SCGM, the PDCGM and the ACCPM adding one column per subproblem at a time.

	SCGM			PDCGM			ACCPM			SCGM/PDCGM		ACCPM/PDCGM	
r	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**or(s)**	**tot(s)**	**ite**	**tot(s)**	**ite**	**tot(s)**
5	27.5	4.7	4.7	11.5	1.5	1.6	22.5	3.1	3.2	2.4	3.0	2.0	2.1
10	32.0	62.7	62.7	15.6	20.4	21.0	29.5	49.1	49.5	2.0	3.0	1.9	2.4
15	38.4	308.8	308.8	20.0	103.8	106.2	36.4	273.2	274.3	1.9	2.9	1.8	2.6
20	45.5	975.6	975.8	25.9	350.5	358.4	42.4	938.7	941.0	1.8	2.7	1.6	2.6

If we consider the average CPU time per iteration for CLSPST modified instances, the PDCGM is the most efficient among the studied strategies, while the SCGM and

the ACCPM have very similar times per iteration. From the evidence gathered so far, one could infer 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 produce important savings in terms of CPU time and column generation iterations.

5.2.4 Performance profiles for large instances

In order to complement our numerical comparisons, we have included performance profiles [31] for each application considering only large instances. To do this, we have removed from this analysis instances considered in the three preceding sections which were small and were efficiently solved by all the strategies. For CSP, instances in class U (Tables 5.1 and 5.2) and the 14 large instances (Table 5.3) were considered. For VRPTW, instances in classes C100, R100, RC100 (Tables 5.4 and 5.5) and the 9 large instances (Table 5.6) were included. Finally, for CLSPST instances in classes G, X1, X2, X3 (Table 5.7) and the replicated instances with r equal to 10, 15 and 20 (Table 5.8) were considered.

In short, performance profiles provide information about the behaviour of different methods for a given metric when solving a set of instances. In our case we are interested in two type of metrics.

When comparing the PDCGM with the SCGM and the ACCPM, we are interested in the number of outer iterations (calls to the subproblem(s)) and the total CPU time needed to solve an instance. For a better understanding of performance profiles, we will briefly describe the methodology proposed in [31].

Having the result for a particular metric (*e.g.,* total CPU time or outer iterations) obtained by using different methods, let us define $\mathcal{M}$ and $\mathcal{I}$ as the set of methods and instances, respectively. Then, for every $i \in \mathcal{I}$ and $m \in \mathcal{M}$, we define $t_{i,m}$ as the result of the metric when solving instance i with method m.

The baseline for every instance will be given by the best result obtained by any of the methods. In our case, and for all metrics considered, this is the minimum of the values among all methods. Therefore, the performance ratio can be defined as:

$$r_{i,m} = \frac{t_{i,m}}{\min\limits_{k \in \mathcal{M}} \{t_{i,k}\}}, \forall i \in \mathcal{I}, \forall m \in \mathcal{M}. \tag{5.5}$$

Additionally, if we define $\mathcal{S}_m(\tau) = \{i \in \mathcal{I} : r_{i,m} \leq \tau\}$, then the cumulative distribution function of method m for the performance ratio is

$$\rho_m(\tau) = \frac{1}{|\mathcal{I}|} |\mathcal{S}_m(\tau)|, \forall m \in \mathcal{M}.$$

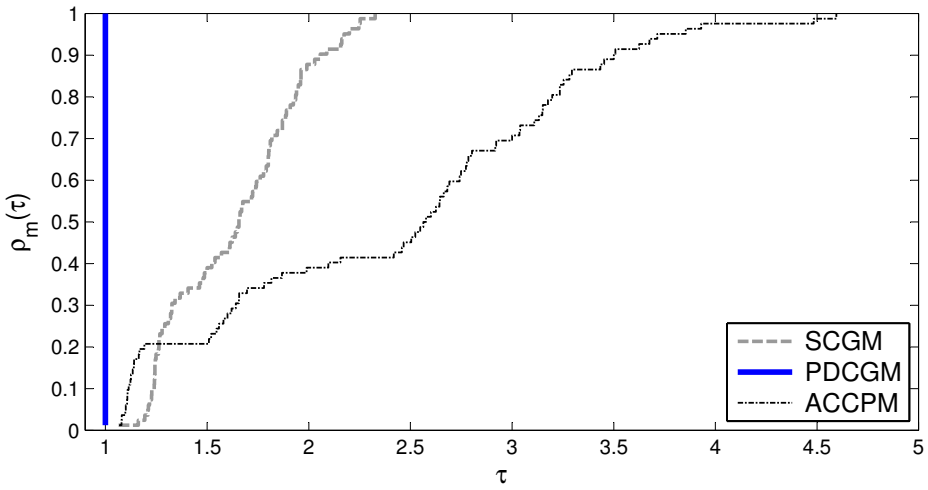
where $\rho_m(\tau)$ represents the probability that the result of method m is between a ratio τ with respect to the best result among all methods. Note that this type of analysis can cope with solver/method failures so it gives a good measure of robustness. Then, in order to generate the cumulative distribution plots of every method for a set of instances (performance profiles), we set several values of τ (x-axis) and plot them against the corresponding $\rho_m(\tau)$ (y-axis). The figures we present in the next section were created with an slightly modified MATLAB [90] script available at `http://www.mcs.anl.gov/~more/cops/`.

Having the results for the aforementioned applications in terms of average and classified by number of columns added per iteration give us a good idea of the efficiency

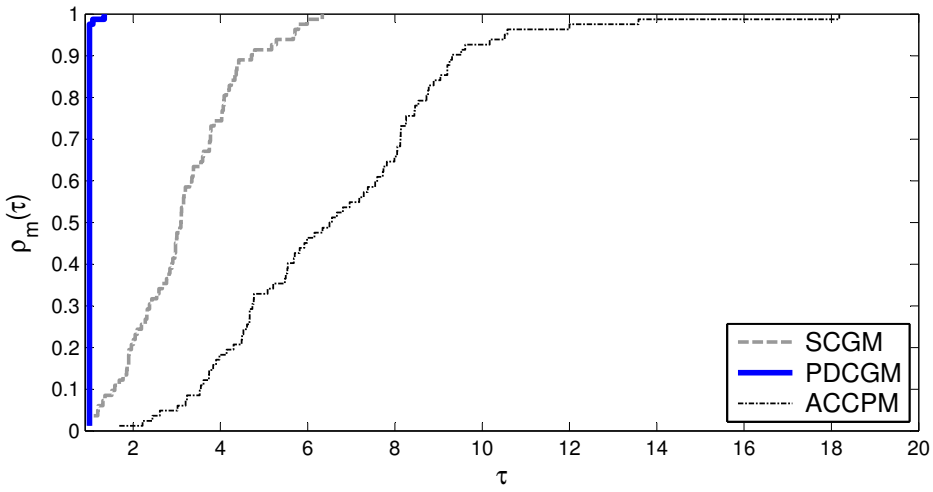
of the methods. It shows clearly that the larger the instances, the better the overall performance of the PDCGM. In order to mitigate the influence of poor performances of some of the strategies in very specific instances (large CPU times or number of outer iterations), we also present the performance profile results for all the difficult instances and all the column strategies. We have selected only instances which challenge the column generation for every application.

Cutting stock problem

In Figures 5.3(a) and 5.3(b) we have the performance profiles in terms of outer iterations and total CPU time, respectively. We have considered instances in class U (Tables 5.1 and 5.2)and the 14 large instances (Table 5.3) and all values of k previously considered.



(a) Outer iterations



(b) Total CPU time

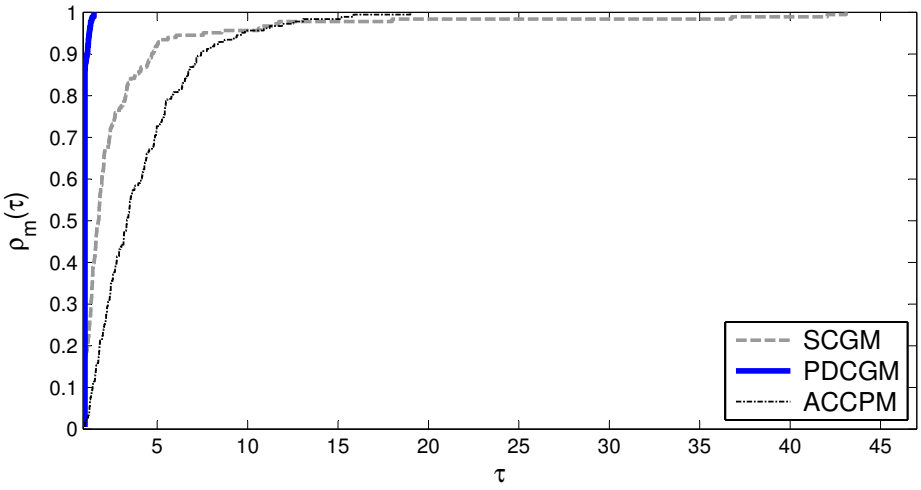
Figure 5.3: Performance profiles for CSP with the the SCGM, the PDCGM and the ACCPM (large instances)

These results clearly complement the discussion in the previous section. It clarifies any doubt and shows that the PDCGM is the best method to solve large instances for

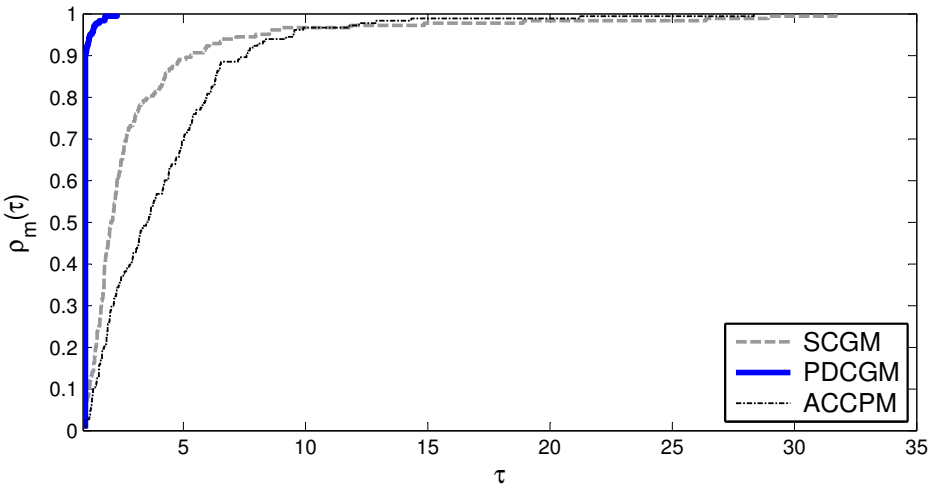
this class of problems. Independently of the number of columns added per iteration and instances considered, the PDCGM always requires fewer outer iterations than the SCGM and the ACCPM as shown in Figure 5.3(a). Also, in terms of CPU time the PDCGM is the most efficient technique for the vast majority of larger instances and when it is not the best, it does not perform badly (only at a factor less than 1.5 from the best strategy). One can observe that the ACCPM is not very competitive and that more than 50% of the instances require at least 6 times more CPU time to be solved than the best result obtained with either the PDCGM or the SCGM.

Vehicle routing problem with time windows

For VRPTW, we have considered instances in classes C100, R100 and RC100 (Tables 5.4 and 5.5) and the 9 large instances with 200, 400 and 600 customers (Table 5.6). In Figures 5.4(a) and 5.4(b) we present the performance profiles for outer iterations and total CPU time, respectively obtained by the three column generation strategies.



(a) Outer iterations



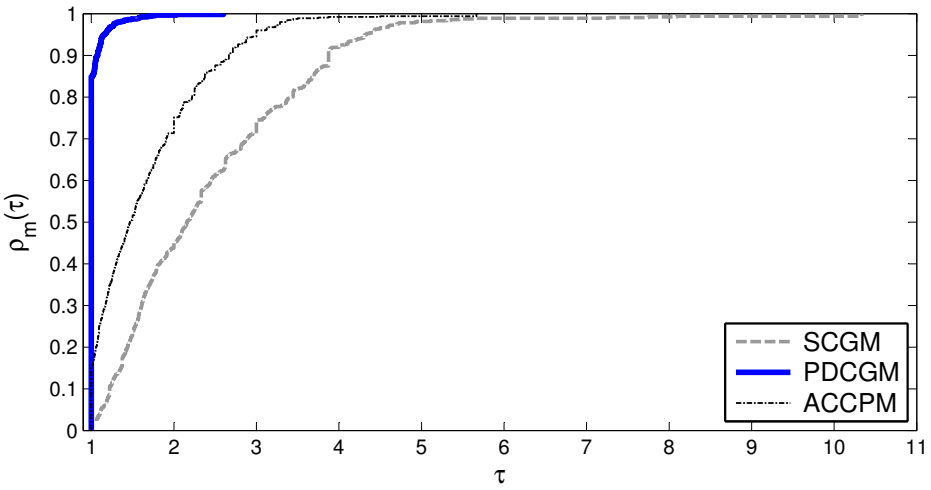
(b) Total CPU time

Figure 5.4: Performance profiles for VRPTW with the SCGM, the PDCGM and the ACCPM (large instances)

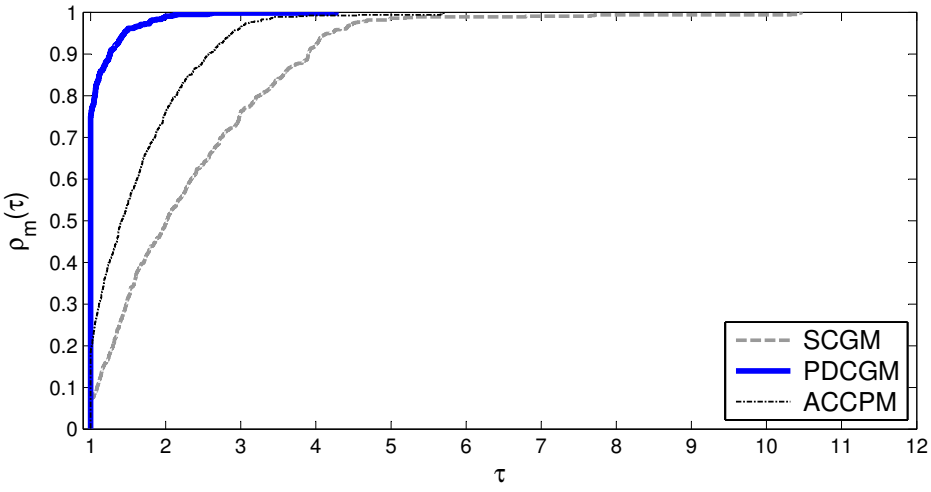
Similar to the results for CSP considering large instances, the PDCGM is the most efficient method among the three approaches considered. It performs consistently better than the SCGM and the ACCPM in more than 90% of the instances in both performance measures.

Capacitated lot-sizing problem with setup times

For the perfomance profiles of CLSPST, we have considered instances in classes G, X1, X2 and X3 (Table 5.7). Additionally, we have included the replicated instances with r equal to 10, 15 and 20 (Table 5.8). In Figures 5.5(a) and 5.5(b) we have the performance profiles for outer iterations and total CPU time, respectively.



(a) Outer iterations



(b) Total CPU time

Figure 5.5: Performance profiles for CLSPST with the SCGM, the PDCGM and the ACCPM (large instances)

Again, the PDCGM is the strategy that obtains the best results for most of the selected instances. It is the most efficient strategy in terms of CPU time in more than 75% of the cases and when it is not the best, it does not perform poorly compared to

the other two strategies. Differently to the results with other two applications, namely CSP and VRPTW, the ACCPM performs much better than the SCGM and it offers the best performance in terms of CPU time in almost 20% of the instances.

With these additional comparisons we aim to provide the reader with enough evidence to support our conclusion that the PDCGM is the variant with the best overall performance among the ones considered in this study when large instances in the context of integer programming are considered.

5.2.5 Additional comments about stabilized column generation

Although no computational study has been performed considering an artificially stabilized column generation based on simplex, we would like to refer to two papers which provide us with some notion on how much can be gained by stabilizing the applications considered in this thesis.

In Rousseau *et al.* [109], they propose an interior point column generation technique in which the dual solutions are convex combinations of extreme dual points of the RMP. These solutions are obtained by modifying the dual objective function randomly. To analyse the computational performance of their approach, Rousseau *et al.* have used the set of large VRPTW instances with 100 customers, namely instances included in classes C100, R100 and RC100. Only the results of 22 out of 29 instances are available. Their comparison involves the implementations of the standard column generation as well as the BoxPen stabilization technique [32]. Since a different subproblem solver, another version of CPLEX and a different machine have been used in their computational experiments, it would not be fair to make a straightforward comparison of the figures presented in their tables with those presented in Tables 5.4 and 5.5. Hence, we have considered the gains 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 time is negligible when compared with the standard column generation. The interior point stabilization (IPS) proposed in [109] shows a better performance than the BoxPen stabilization, being 1.4 times faster than the standard column generation technique. For the same set of instances (22 out of 29 instances, with $n = 100$), the PDCGM is 2 times faster than the SCGM on average ($k = 200$).

In Briant *et al.* [17], a thorough computational study is presented. They compare the standard column generation (Kelley cutting plane method) with the bundle method which is a stabilized cutting plane method which uses quadratic stabilization terms as shown in Section 3.5. In the computational experiments in [17] results for CSP and CLSPST (MILS problem in their paper) are included. Their results for CSP, indicate that using the bundle method may slightly reduce the number of column generation iterations at the cost of worsening the total CPU time by a factor greater than 3 (see Tables 1 and 2 in [17]). Furthermore, in the results obtained for CLSPST, the bundle method shows a poor behaviour in terms of the number of outer iterations and CPU time when compared with the standard column generation (see Table 12 in [17]). In our computational experiments, and considering these two applications, namely CSP and CLSPST, the overall results obtained with the PDCGM outperform the ones obtained with the SCGM.

Two final comments with regard to the ACCPM are needed. Looking at our computational results and considering all the applications studied in this thesis, it seems that the ACCPM suffers when multiple columns are added at each iteration of the column generation procedure. As mentioned in [9], the starting point is critical for the success

of the ACCPM [9] in the context of column generation and Lagrangian relaxation for integer programming problems.

Initializing the ACCPM with poor columns may originate unnecessary iterations at the beginning of the column generation process, which is expensive. This may be emphasised by the addition of unnecessary columns at every iteration making the method converge slowly. A remedy to this could be to choose carefully the initial point for each instance. However, this is impractical and beyond the scope of this study. Also, in order to guarantee a good performance of the ACCPM so it can efficiently reoptimize after new columns are added, the theory requires some safeguards. For instance, if multiple columns are added in one iteration and the old analytic centre deeply violates these new constraints in the dual space, theoretically the ACCPM struggles and no warmstarting is possible [51, 52].

As a final remark of this chapter, from our results one can observe that unlike the SCGM, the strategies based on interior point methods, namely the PDCGM and the ACCPM, spend a non-negligible amount of time solving the RMPs. This is because reoptimizing interior point methods is not as straightforward as reoptimizing with simplex-type methods. This issue will be extensively discussed in the next chapter.

Chapter 6

A New Reoptimization Technique for the Primal-Dual Column Generation Method

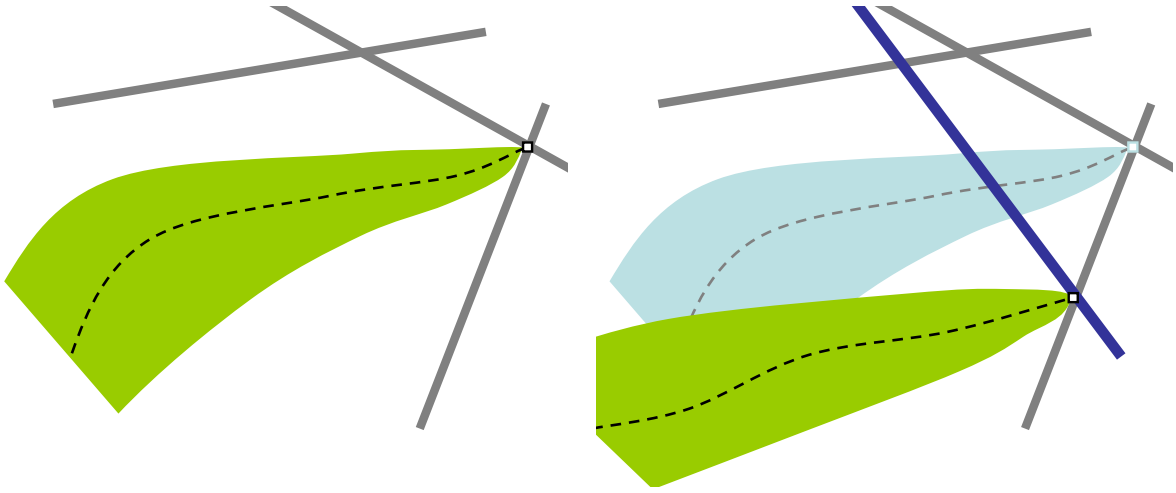
In this chapter we present a new warmstarting technique applicable in the context of the primal-dual column generation method described in Chapter 5. We first review some warmstarting techniques for interior point methods. Then, a thorough theoretical analysis of a new approach is performed and computational experiments are presented to show its behaviour in practice. This chapter follows closely the developments and results available in [59].

A warmstarting strategy is understood as the use of previous information gathered in the solution process of a given problem in order to solve a subsequent related problem. The aim of such a strategy is to ease the solution process for the modified problem. In contrast, a coldstart approach is when no prior information is used and the modified problem is solved from scratch. The difficulty of interior point methods to reoptimize when compared to active set methods is a well-known issue (see for instance [13]).

Finding a warmstarting point with a simplex-type method can be easily done if the optimal basis of the previous problem remains either primal feasible or dual feasible so the new problem can efficiently be solved by the primal or the dual simplex method, respectively. In the case of column generation, when new columns are added to the RMP, the optimal solution of the previous RMP can be used as a basis for the new RMP (the variables corresponding to the new columns are set to zero).

The difficulties reoptimizing interior point methods result from the way these methods approach optimality. To guarantee fast convergence IPMs work in the interior of the feasible set (by exploiting the notion of central path and forcing the iterates to stay in its neighbourhood) and approach the boundary of the feasible region only close to termination. Keeping the iterates in the interior of the feasible set is the great advantage of IPMs and responsible for their spectacular efficiency [57, 121]. However, it becomes a problem if one tries a naive warmstarting with these methods. In general IPMs should not use the *optimal* solution of the previous problem as initial iterate to reoptimize because such point may be close to the boundary of the feasible set and is very likely to be far away from the new central path [56].

An illustration of this situation and how the central path changes after the addition of one constraint in the dual space is depicted in Figure 6.1.



(a) RMP^k (b) RMP^{k+1}

Figure 6.1: Differences between two consecutive RMPs - Changes in the central path and its neighbourhood

To overcome these difficulties, several reoptimization strategies have been proposed in the literature which are described in the next section.

6.1 Warmstarting strategies

There have been several solutions proposed to overcome the difficulties of warmstarting IPMs. They essentially differ and are specialised for two different situations: one in which the problem size remains the same but the problem data changes and one in which both the problem size and its data change.

In [44], Freund has proposed a shifted barrier method for solving linear programs using an infeasible warmstart. Freund has shown that under suitable assumptions his potential function reduction algorithm runs in polynomial time. In the same spirit, Benson and Shanno [13] have presented an exact primal-dual penalty approach. They have proposed a method which relaxes the non-negativity constraints of primal and dual problems and penalizes any violation of them. This method accounts for small changes in the data and requires the setting of some penalization parameters which may be non-trivial. Engau *et al.* [35] have reformulated the primal-dual pair using a slacked form (without penalization terms). Theoretical guarantees under which warm-started IPMs outperform coldstart IPMs are presented. Engau *et al.* have tested their approach considering changes in the data (linear programming) as well as in a cutting plane context (combinatorial optimization). One of the most remarkable features of this strategy is that it does not require extensive setting of parameters and is readily applicable to a wide variety of problems. All the aforementioned methods, [13, 35, 44] modify the original problem, adding auxiliary variables to deal with infeasible initial points. Then these methods drive these auxiliary variables to zero, obtaining the optimal solution of the original problem.

A different class of warmstarting methods calculates a warmstarting point and reoptimizes the modified problem from this point without adding extra parameters.

In [93], Mitchell has studied different strategies to obtain an interior point when adding constraints (cutting planes) and variables to a linear programming problem. These methods rely on calculating directions based on the projection onto the null

space of the constraint matrix. A step towards these directions is always possible so the methods recover feasibility in one step. The methods have been implemented using the primal projective standard-form variant of Karmarkar's algorithm for linear programming within a cutting plane method applied to matching problems [93, 97]. Additionally, Mitchell and Borchers [96] have studied a different and practical warmstarting strategy using the primal-dual barrier method applied to solve linear ordering problems.

Goffin and Vial in [51] and [52] have studied how to deal with the addition of single and multiple cuts in a cutting plane framework, respectively. They have proposed a way of restarting the analytic centre cutting plane method [49] already described in Chapter 3. The strategy assumes that the cuts are central using a Dikin's ellipsoid as reference. In order to determine the direction that recovers feasibility, the method requires a non-linear optimization problem to be solved. Goffin and Vial have provided bounds on the number of steps to recover an analytic centre which depends on the number of cuts added. Similarly, Oskoorouchi *et al.* [103] propose a method that recovers dual feasibility in a cutting plane framework under the condition that the constraints added are moderately deep. As stated in [51, 52], if there is at least one deep constraint (constraint which does not intersect the Dikin's ellipsoid around the query point), the proposed method recovers only primal feasibility.

A different approach has been presented by Gondzio [56] in a primal-dual cutting plane method context. He has proposed fixing the initial values of the new primal and dual variables and then restoring primal and dual feasibilities independently. To calculate the warmstarting point, modified Newton steps have to be performed. Gondzio has distinguished between deep and shallow cuts, adjusting his strategy accordingly. Although a successful implementation has been developed, *no theoretical guarantee for its performance has been provided.*

In [60], Gondzio and Grothey introduced a primal-dual interior point method that relies on multiple-centrality corrector techniques to find a warmstarting solution. From this point their method seeks feasibility. They have shown the conditions under which their short and long step path following methods can absorb data perturbations in one step or a few steps (for larger perturbations). They have extended their analysis to problems with special structures (primal and dual block angular structures). Gondzio and Grothey have also studied an unblocking strategy based on sensitivity analysis [61]. The proposed method deals with the blocking issue that any advanced starting point may suffer from by increasing the step size allowed in the Newton direction. First, their strategy identifies which components the blocking originates from. Then, the method aims to remove the blocking components using sensitivity analysis. Conditions that an unblocking direction must satisfy and proofs of the existence of such direction have been provided. They have observed that a large value of the duality measure can absorb more infeasibility in one step than a small value. Additionally, a well-centred point can allow an IPM to absorb more infeasibility.

Yildirim and Wright [123] have studied different criteria to determine a well-suited warmstarting point. They have compared several different strategies of direction searching (*i.e.,* (weighted) least-square and Newton approaches) and given complexity and convergence results for each of these methods after changes in the data. They have determined the size of the perturbation that can be absorbed by each of these methods. Numerical experiments and further developments in this direction have been presented by John and Yildirim in [74].

Fliege in [40] introduces a new approach to solve convex multicriteria optimization problems where changes in the objective function coefficients are common. Having two

closely related problems, the method applies few centering steps to the old iterate, so the new point is very close to the new central path. He shows that to compute a finite set of discrete points to describe the solution set, his method runs in polynomial time.

A very different warmstarting approach has recently been proposed by Skajaa *et al.* [111]. In that paper, they employ a self-dual embedding linear programming model which facilitates taking any point as a warmstarting candidate including points close to the boundary of the feasible region.

Finally, in [35, 56] computational experiments demonstrate how efficient warmstarting methods are when compared to coldstart in a cutting plane framework.

6.2 Theoretical developments

In this section we introduce a specialized warmstarting strategy applicable within the primal-dual column generation scheme described in Chapter 5. It is specialized because we take advantage of some structure/properties which are often present when solving combinatorial optimization problems. We have observed that in many applications all the elements in A of every RMP are non-negative. For instance, columns in the primal space describing cutting patterns (CSP), routes (VRPTW) or production plans (CLSPST) by definition contain only non-negative entries. In our developments we do consider this observation and therefore, the analysis and results should be understood in this context.

Before describing the new strategy, let us recall the notation introduced in Chapter 2 and assume that we are in an intermediate iteration of the column generation process. The current RMP (see (2.1)) characterized by A, b, c has been solved and the oracle generates k new columns with parameters $(\bar{A}, \bar{c})$. Note that $\bar{c}_j - \bar{A}_j^T y < 0$ for all $j \in K = \{1, \ldots, k\}$ for a dual solution y of the current RMP (recall that we only add columns with negative reduced costs). Hence, the modified primal-dual pair can be written as

$$\mathcal{P}_1 := \min \quad c^T x + \bar{c}^T \bar{x}, \quad \text{s.t.} \quad Ax + \bar{A}\bar{x} = b, \quad x \geq 0, \quad \bar{x} \geq 0, \tag{6.1a}$$

$$\mathcal{D}_1 := \max \quad b^T y, \quad \text{s.t.} \quad A^T y + s = c, \quad \bar{A}^T y + \bar{s} = \bar{c}, \quad s \geq 0, \quad \bar{s} \geq 0, \tag{6.1b}$$

where k is the number of columns (variables) added to the original problem (2.1), $\bar{x} \in \mathbb{R}^k$ is the vector of new primal variables and $\bar{s} \in \mathbb{R}^k$ is the vector of new dual slack variables. $\bar{A} \in \mathbb{R}^{m \times k}$ represents the coefficient matrix for the new variables and $\bar{c} \in \mathbb{R}^k$ the vector of objective function coefficients. Note that new variables in the primal space are associated with new constraints in the dual space. As we associate variables with columns, constraints in the dual space can be interpreted as the cuts which restrict the dual localization set. From now on we will use the terms columns and cuts interchangeably, where the former term refers to the primal space and the latter to the dual. This will allow us to describe some concepts and correspondence with other strategies.

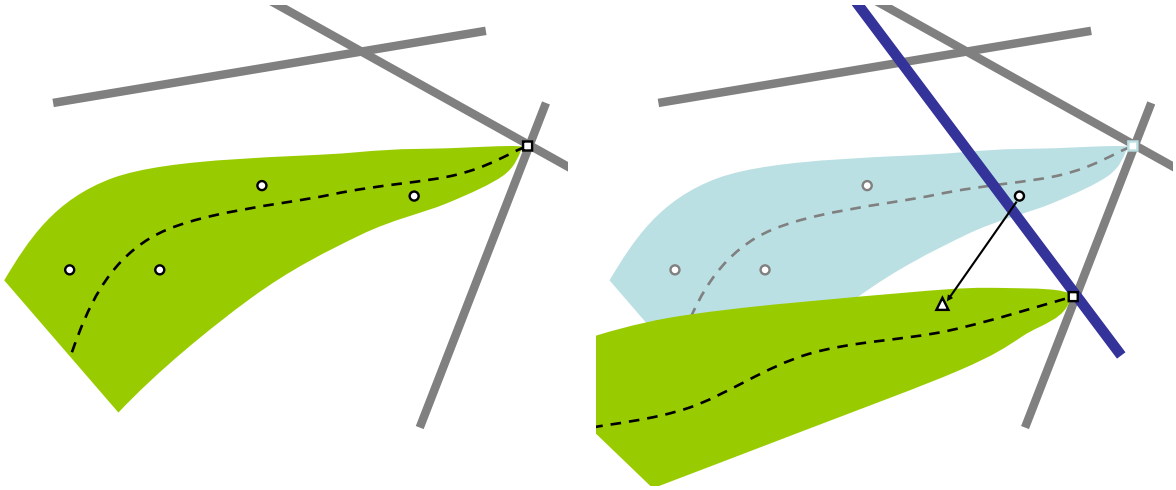
Similar to system (2.7) obtained after applying the first order optimality conditions to (6.1), the new Newton-like system of equations is

$$\begin{bmatrix} A & \bar{A} & 0 & 0 & 0 \\ 0 & 0 & A^T & I & 0 \\ 0 & 0 & \bar{A}^T & 0 & I \\ S & 0 & 0 & X & 0 \\ 0 & \bar{S} & 0 & 0 & \bar{X} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \bar{x} \\ \Delta y \\ \Delta s \\ \Delta \bar{s} \end{bmatrix} = \begin{bmatrix} \xi_{\bar{b}} \\ \xi_c \\ \xi_{\bar{c}} \\ \tau\mu e - XSe \\ \tau\mu e - \bar{X}\bar{S}e \end{bmatrix}, \tag{6.2}$$

where $\xi_{\bar{b}} = b - Ax - \bar{A}\bar{x}$, $\xi_{\bar{c}} = \bar{c} - \bar{A}^T y - \bar{s}$, $\bar{X} = diag\{\bar{x}_1, \ldots, \bar{x}_k\}$, $\bar{S} = diag\{\bar{s}_1, \ldots, \bar{s}_k\}$, $\mu = (x^T s + \bar{x}^T \bar{s})/(n+k)$ and $\tau \in (0,1)$ is the centering parameter.

The process of finding a warmstarting point has been divided into two stages. In the first stage, we need to find a point from a list of stored iterates which, in the dual space, is not too deeply violated by the added cuts. The conditions to determine this point are discussed in Section 6.2.4. In the other stage, an adjustment $(\Delta x, \Delta y, \Delta s)$ has to be computed taking this point as a reference. After this, a full step is taken in this direction and the old point is updated in the new dimensions to produce a full-dimension warmstarting point. From this point we continue iterating and solving system (6.2). Our aim is to find a starting point which reduces the number of iterations required to solve problem (6.1) when compared to a coldstart approach. To develop such a two-stage approach we need to deal with two problems: (a) feasibility in the primal and dual spaces and (b) centrality of the new warmstarting point. We address both issues later.

An illustration of this strategy can be found in Figure 6.2 where the circles indicate stored iterates.



(a) RMP^k (b) RMP^{k+1}

Figure 6.2: The one step primal-dual restoration strategy

Since we are designing a new warmstarting technique to be applied in the PDCGM, it is sensible to point out the similarities and main differences with the strategy currently in use [56] and other strategies presented in the literature such as the ones studied in [123] and [52].

Despite the fact that our developments follow the idea in [56] of treating primal and dual infeasibilities separately, our approach differs in two key aspects. Firstly, in the choice of $\bar{x}$ and $\bar{s}$. The aforementioned paper sets

$$\bar{s}_j = \max\left\{|\bar{c}_j - \bar{A}_j^T y|, \mu^{1/2}\right\}, \tag{6.3}$$

for every added cut j. Since it is likely to have (deep) cuts where $\bar{c}_j - \bar{A}_j^T y \ll 0$, $\bar{s}_j$ usually takes large values. Furthermore, since the complementarity products of the new variables are set to $\bar{x}_j \bar{s}_j = \mu$, for every $j = 1, 2, \ldots, k$, such strategy is likely to produce small values of $\bar{x}_j$. This goes against the expectations that the new variables corresponding to the recently appended (deep) cuts are likely to take non-zero values at

the optimal solution. In our strategy we do not choose $\bar{x}_j$ as a function of $\bar{s}_j$. Instead, we choose these values bearing in mind their impact on centrality.

A second difference between the strategy of [56] and the one proposed in this thesis is in the definition of the search direction to find the new warmstarting point. In the method proposed in [56], the search direction is calculated using a variation of system (6.2) via target $\bar{t}$ as shown in (6.4), which is a consequence of the choice of $\bar{x}_j$ and $\bar{s}_j$. The system to be solved is

$$\begin{bmatrix} A & \bar{A} & 0 & 0 & 0 & 0 \\ 0 & 0 & A^T & I & 0 & 0 \\ 0 & 0 & \bar{A}^T & 0 & I & 0 \\ S & 0 & 0 & X & 0 & 0 \\ 0 & \bar{S} & 0 & 0 & \bar{X} & -I \\ 0 & 0 & 0 & 0 & I & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \bar{x} \\ \Delta y \\ \Delta s \\ \Delta \bar{s} \\ \bar{t} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ z - \bar{s} \\ 0 \\ \bar{X}\bar{S}e \\ -\bar{s} \end{bmatrix}, \tag{6.4}$$

where $z_j = \bar{A}_j^T y_0 - \bar{c}_j$. Solving (6.4) gives $\bar{t} = -\bar{S}(\bar{A}^T(A\Theta A^T)^{-1}\bar{A})^{-1}z$. In the dual space, the method ensures that after a full step is taken in this direction, dual feasibility is recovered in the new components. However, this method does not guarantee the same for the old components since some blocking in small components may occur. In the primal space, feasibility can relatively easily be restored by choosing small $\bar{x}_j$'s. It is accepted that the complementarity products for the old variables may get significantly worse as a price to pay for recovering primal and dual feasibilities. In our proposed strategy, we aim to restore primal and dual feasibilities using auxiliary linear optimization problems. The principal aim of both, primal and dual auxiliary problems, is to seek feasibility while minimizing changes in those old variables which are small. The rationale is to minimize changes in the complementarity products.

Our approach follows a variation of a weighted least-square strategy proposed in [123]. It differs from [123] in the sense that we only consider small variables and that we have extra constraints to satisfy in both primal and dual spaces.

Additionally, the method presented in this chapter has several other different characteristics when compared with the strategy proposed in [52]. Firstly, in [52, 103], the strategy recovers primal and dual feasibility by solving a non-linear problem while in our approach we aim to minimize large variations in those old components which are small using linear functions and extra constraints. Secondly, the strategies proposed in [52, 103] work very well if all the new constraints traverse the Dikin's ellipsoid around the query point while our approach relies on a different assumption which is that the depth of the added constraints does not exceed a reference value which is a function of μ and γ. Thirdly, we are able to retreat back in the list of iterates, while the strategies in [52, 103] take the last iterate as the reference point and adjust it. In spirit, the strategies developed for the analytic centre cutting plane method [52, 103] and the one described in this chapter have the same objective which is to obtain a well-centred warmstarted iterate after adding new columns/constraints. However, while our technique relies on the notion of the central path, the other approaches aim to get a point close to the new analytic centre.

In what follows in this chapter, we analyse the restoration of primal and dual feasibilities, the consequences for the new complementarity products and describe the complete algorithm. At the end, we provide computational evidence that the method compares favourably with coldstart.

6.2.1 Technical conditions

Let (x^0, y^0, s^0) be a feasible solution of the primal-dual pair (2.1). Also, let us recall the primal-dual strictly feasible set and the symmetric neighbourhood of the central path defined in Chapter 2 as

$$\mathcal{F}^0 = \{(x, y, s) : Ax = b; A^T y + s = c; (x, s) > 0\},$$
$$\mathcal{N}_s(\gamma) = \{(x, y, s) \in \mathcal{F}^0 : \gamma\mu \leq x_j s_j \leq \frac{1}{\gamma}\mu; \forall j = 1, 2, ..., n\}.$$

We are interested in $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$ for a fixed $\gamma \in (0, 1)$. The reader must note that our definition of the neighbourhood is a slight modification of the wide neighbourhood $\mathcal{N}_{-\infty}(\gamma)$ in [121] proposed in [60]. The aim of our specialized warmstarting algorithm is to find an initial point from which to start solving the modified problem (6.1). This initial point is obtained from information gathered when solving problem (2.1) via system of equations (2.7). Note that $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$, implies $\xi_b = \xi_c = 0$.

Before continuing, let $N = \{1, \ldots, n\}$, $M = \{1, \ldots, m\}$ and $K = \{1, \ldots, k\}$ be index sets. Additionally, let us state the following general assumptions which are technical conditions we will use in our theoretical results in the following sections.

(A.1) All elements in matrices A and $\bar{A}$ are non-negative.

(A.2) We are able to store a list of iterates which are strictly feasible and well-centred (*i.e.*, $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$).

(A.3) There exists a $\mathcal{U}_v$ such as $\|x^0, s^0\|_\infty \leq \mathcal{U}_v$.

(A.4) $\mathcal{U}_v^2 > 1$.

Assumption (A.1) is motivated by the applications we are interested in solving. There are several examples in combinatorial optimization where this assumption is valid. We refer the reader to Chapter 4 to see three applications in which this condition holds. Additionally, there is a wide variety of problems which satisfy this condition. See for instance the travelling salesman problem in [17] and the references therein, or the uncapacitated facility location problems [22], just to mention a few of them. Although this assumption is not valid for general combinatorial optimization problems, it covers a wide variety of applications.

Assumption (A.2) is easily met if the feasible path following algorithm is used to solve the problem.

Assumption (A.3) is a technical requirement similar to the one used in [123] which bounds the largest primal and dual slack variables. Note that in [123, Theorem 2.1.], and following the analysis in [102], Yildirim and Wright have shown that for a strictly feasible point (x, y, s) which does not necessarily lie exactly on the central path but belongs to its neighbourhood, the following two bounds are satisfied

$$\|x\|_2 \leq \mathcal{C}(d)\left(\mathcal{C}(d) + \frac{\mu n}{\|d\|_2}\right),$$
$$\|s\|_2 \leq 2\|d\|_2\,\mathcal{C}(d)\left(\mathcal{C}(d) + \frac{\mu n}{\|d\|_2}\right),$$

where d represents a triplet containing the instance data ($d = (A, b, c)$), and $\mathcal{C}(d)$ is the condition number [102]. Similarly, in our case it is easy to see that $\mathcal{C}(d) < \infty$

(feasible RMPs) and since we use a path-following method and a minor variation of the neighbourhoods used in [123], Assumption (A.3) is satisfied.

Assumption (A.4) is also a technical condition which can be easily imposed by setting $\mathcal{U}_v = \max\{\|x^0, s^0\|_\infty; 1+\epsilon\}$, where $\epsilon = 10^{-8}$.

In addition to these assumptions and due to the nature of our applications, we can always initialize the column generation procedure with some initial columns so the initial RMP is feasible (for instance, see the initializations used in Chapter 5).

Using our definition of the central path neighbourhood for $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$, we have

$$\gamma\mu_0 \leq x_j^0 s_j^0 \leq \frac{1}{\gamma}\mu_0 \quad \forall j \in N. \tag{6.5}$$

Both this definition and the upper bound $\mathcal{U}_v$ allow us to bound x_j^0 and s_j^0, for every $j \in N$, as follows

$$\frac{\gamma\mu_0}{\mathcal{U}_v} \leq \quad x_j^0 \quad \leq \quad \mathcal{U}_v, \tag{6.6a}$$

$$\frac{\gamma\mu_0}{\mathcal{U}_v} \leq \quad s_j^0 \quad \leq \quad \mathcal{U}_v. \tag{6.6b}$$

The similarities between the initial RMP and the modified RMP are one of the key elements to take into consideration when a warmstarting strategy is designed for an interior point method. If there is no similarity between the initial and modified problems, we can expect that a warmstarting strategy will not lead to any substantial improvement when compared with a coldstart approach. This could be the case when the new RMP has a completely different feasible region around the optimal solution and a close-to-optimality solution deeply violates the new constraints. Therefore, any information previously gathered close to the optimal solution will not help to speed up the solution process of the modified problem. Then, there is a need to understand the relation between the added cuts and the previous RMP. A sensible way to proceed is to measure the size of the newly added cuts in terms of the current penalty parameter, μ_0. Let the inequality $-(\bar{c}_j - \bar{A}_j{}^T y^0) \leq f(\mu_0)$ be used to determine the depth of the cuts, where $f(\mu_0)$ is an increasing function of μ_0. Note that by using this idea of associating the depth of the cuts to μ_0 and having a list of iterates, we could retreat far enough in the iteration process to make the cuts relatively shallower. In other words, for all the cuts we could choose a suitably large μ_0 to decrease their relative depth.

As mentioned earlier, the choices of $\bar{x}$ and $\bar{s}$ have important consequences in the primal-dual infeasibilities and in the complementarity conditions. We aim to find a warmstart which: (a) is feasible in the primal and dual space; and (b) keeps the complementarity products reasonably small and inside a slightly modified neighbourhood if the cuts satisfy some desirable properties. We expect the duality gap to increase since we are adding variables/constraints to the primal/dual problem. However, we would like to keep this duality measure relatively close to the old one.

Now, let us define some sets and parameters used throughout the chapter.

Definition 6.1. *Let $\mathcal{B}_0$ be the set containing all indices j such that $x_j^0 \geq s_j^0$, where $j \in N$. We call this set the* primal dominant partition *at solution (x^0, y^0, s^0). Conversely, the* dual dominant partition *at solution (x^0, y^0, s^0) is defined as $\mathcal{N}_0$ and contains all indices j such that $x_j^0 < s_j^0$, where $j \in N$. Clearly $\mathcal{B}_0 \cup \mathcal{N}_0 = N$ and $\mathcal{B}_0 \cap \mathcal{N}_0 = \emptyset$.*

The reader familiar with the simplex method may be tempted to interpret the sets $\mathcal{B}_0$ and $\mathcal{N}_0$ as a guess of the basic-nonbasic partition. In the spirit of IPM, these sets

provide only an early guess of the primal-dual strictly complementarity partition which in general is not equivalent to the basic-nonbasic partition.

Definition 6.2. *Considering $A \in \mathbb{R}_+^{m\times n}$, let us define*

$$A_{min} := \min_{i\in M, j\in N: A_{ij}>0} \{A_{ij}\},$$

as the minimum non-zero element of matrix A. Similarly,

$$a_{min} := \min_{i\in M: a_i>0} \{a_i\},$$

is the minimum non-zero element of vector $a \in \mathbb{R}_+^m$. Additionally, $\sigma_{max}(A)$ and $\sigma_{min}(A)$ denote the maximum and minimum singular value of matrix A, respectively and

$$\sigma_{max} = \max\{\sigma_{max}(A), \sigma_{max}(\bar{A})\}.$$

In the following three sections we will prove the main results of this chapter. Namely, we will provide the methodology to choose a good warmstart solution such that the primal and dual feasibilities may be restored in one Newton step and will show that this can be achieved without significantly affecting the proximity of the new iterate to the new central path.

6.2.2 Dual feasibility restoration

In order to minimize the impact of restoring dual feasibility and to measure its effect on complementarity products, we have defined an auxiliary linear optimization problem. Taking the second and third equations of (6.2) and considering $(x^0, y^0, s^0) \in \mathcal{F}^0$, we have

$$\begin{aligned} A^T\Delta y + \Delta s &= 0, && (6.7)\\ \bar{A}^T\Delta y + \Delta\bar{s} &= \bar{c} - \bar{A}^T y^0 - \bar{s}. && (6.8) \end{aligned}$$

Such a formulation allows for a considerable freedom in the choice of Δy. This system of equations is likely to have multiple solutions.

The following auxiliary linear optimization problem minimizes the relative change of variables in the primal dominant partition (*i.e.*, corresponding to small s_j^0) when a solution of problem (2.1) is available.

$$\begin{aligned} \text{minimize} \quad & \sum_{j\in\mathcal{B}_0} \frac{\Delta s_j}{s_j^0}, && && (6.9a)\\ \text{subject to} \quad & \sum_{i\in M} \bar{A}_{ij}\Delta y_i + \Delta\bar{s}_j = \bar{c}_j - \sum_{i\in M} \bar{A}_{ij} y_i^0 - \bar{s}_j, && \forall j \in K, && (6.9b)\\ & -\sum_{i\in M} A_{ij}\Delta y_i = \Delta s_j, && \forall j \in \mathcal{B}_0, && (6.9c)\\ & \Delta y_i \le 0, && \forall i \in M, && (6.9d)\\ & \bar{s}_j^w \ge \max\left\{\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}, \bar{c}_j - \bar{A}_j^T y^0\right\}, && \forall j \in K, && (6.9e) \end{aligned}$$

where $\bar{s}_j^w = \Delta\bar{s}_j + \bar{s}_j$, for every $j \in K$. Observe that by solving this linear problem,

feasibility for the new components is achieved via constraints (6.9b) (the third equation of system (6.2)). Note that the aim of this linear program is to minimize the relative change of small slacks of the dual problem while a full step is feasible for all components. The motivation behind this objective is that we would like to perturb the previous solution as little as possible. Constraints (6.9e) ensure that the new slack variables are bounded away from zero. Although one would expect to have

$$\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}} > 0 > \bar{c}_j - \bar{A}_j^T y^0, \quad \forall j \in K,$$

this is not always the case due to the backtracking feature of our strategy. This will be explained in more detail later in this chapter.

The linear problem (6.9) can be simplified by the following steps. First, let us eliminate Δs_j for every $j \in \mathcal{B}_0$ by using constraints (6.9c) and substituting $\Delta y = -\Delta\bar{y}$. Also, let us introduce a new parameter f_i for every $i \in M$ such that $f_i = \sum_{j\in\mathcal{B}_0} \frac{A_{ij}}{s_j^0}$. Additionally, let us set

$$\bar{s}_j = \max\left\{\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}, \bar{c}_j - \bar{A}_j^T y^0\right\}, \quad \forall j \in K. \tag{6.10}$$

Now, we can rewrite problem (6.9) as

$$\begin{aligned}
\mathcal{D}_{aux} := \text{minimize} \quad & \sum_{i\in M} f_i \Delta\bar{y}_i, && && (6.11a)\\
\text{subject to} \quad & \sum_{i\in M} \bar{A}_{ij}\Delta\bar{y}_i - \Delta\bar{s}_j = -(\bar{c}_j - \sum_{i\in M}\bar{A}_{ij}y_i^0 - \bar{s}_j), && \forall j \in K, && (6.11b)\\
& \Delta\bar{y}_i \geq 0, && \forall i \in M, && (6.11c)\\
& \Delta\bar{s}_j \geq 0, && \forall j \in K. && (6.11d)
\end{aligned}$$

To avoid unbounded solutions, if $f_i = 0$ for a given $i \in M$, we set $f_i = 1$. Observe then that by assumption (A.1), $f_i > 0$ for every $i \in M$. Therefore, the problem (6.11) can be interpreted as finding the minimum adjustment $\Delta\bar{y}$ of dual variables y such that at the new point $y + \Delta y$ $(= y - \Delta\bar{y})$ all dual feasibility constraints in (6.1b), including the ones corresponding to new deep cuts, are satisfied. The objective function (6.11a) promotes such changes $\Delta\bar{y}$ which do not alter the primal dominant components of s^0 (the components corresponding to small values s_j^0). The purpose of the next lemma is to show that the problem $\mathcal{D}_{aux}$ has a bounded solution.

Lemma 6.3. *Given $\bar{c}_j - \bar{A}_j^T y^0$ for every $j \in K$ representing the reduced cost of column j, where $\bar{A}_j$ represents the j-th column of matrix $\bar{A}$, $\bar{A}_{ij} \geq 0$ for every $i \in M, j \in K$, problem* (6.11) *has a bounded solution.*

Proof. From (6.11b) we have

$$\bar{A}^T\Delta\bar{y} - \Delta\bar{s} = -(\bar{c} - \bar{A}^T y^0 - \bar{s}) = d. \tag{6.12}$$

If $\bar{c}_j - \bar{A}_j^T y^0 < \gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}$ and from definition (6.10), clearly $d_j > 0$. When $\bar{c}_j - \bar{A}_j^T y^0 \geq \gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}$, the corresponding $d_j = 0$. Note that we could determine a feasible solution to problem (6.11) in the following way. For every $j \in K$, calculate

$\Upsilon_j = d_j/(\bar{A}_j)_{min}$. Then, define $\Delta\bar{y}_i$ for every $i \in M$ as

$$\Delta\bar{y}_i = \max_{j\in K}\{\Upsilon_j\}. \tag{6.13}$$

This solution is feasible since it will produce $\Delta\bar{s}_j \geq 0$ for every $j \in K$. Also, observe that any value of $\Delta\bar{y}_i$ exceeding (6.13) will not improve the objective function and therefore, equation (6.13) determines an upper bound for the $\Delta\bar{y}$ components. In other words, and for every $i \in M$, $\Delta\bar{y}_i^\star \leq \max_{j\in K}\{\Upsilon_j\}$, where $\Delta\bar{y}_i^\star$ is the optimal solution of problem (6.11). Hence, we can conclude that the problem (6.11) has a bounded solution. □

If $d_j = 0$ for every $j \in K$, the trivial solution $(\Delta\bar{y}, \Delta\bar{s}) = 0$ is the optimal solution. This case is unlikely since we expect columns with negative reduced costs, namely $\bar{c}_j - \bar{A}_j^T y^0 < 0$. However, and as it will be explained later, due to our strategy in some occasions this may happen.

The next lemma states that by calculating direction $\Delta\bar{y}$ from (6.11) infeasibilities are absorbed in the old components and a dual feasible point for the new problem is obtained.

Lemma 6.4. *Let $(x^0, y^0, s^0) \in \mathcal{F}^0$. If $\Delta\bar{y}$ is chosen by solving* (6.11) *then a full step in the old components is feasible. Therefore, dual feasibility is restored in one step.*

Proof. From the definition of problem (6.11), we have $\Delta\bar{y}_i \geq 0$ for every $i \in M$ and therefore, $\Delta y_i \leq 0$ for every $i \in M$. Since $A_{ij} \geq 0$ for every $j \in N$ and $i \in M$ and using (6.7), we have:

$$\Delta s_j = -\sum_{i\in M} A_{ij}\Delta y_i \geq 0, \quad \forall j \in N.$$

Recalling that $s_j^0 > 0$, $s_j^0 + \Delta s_j > 0, \forall j \in N$ as required. □

Even though our objective is to minimize the variation of the dual elements in the primal dominant partition, we cannot guarantee this variation to be small. This is an unavoidable consequence of the fact that there is no control of the depth of the new cuts and a large variation of some components Δs is expected. Despite the lack of control, we can still determine an upper bound for Δs in the old components. Note that from (6.7) and $\Delta y = -\Delta\bar{y}$, we have

$$\Delta s = A^T \Delta\bar{y}, \tag{6.14}$$

Also, from (6.13) we have the following bound on $\Delta\bar{y}_i$ for every $i \in M$

$$\Delta\bar{y}_i \leq \frac{d_{max}}{\bar{A}_{min}}, \tag{6.15}$$

where, following (6.12), d_{max} is defined as

$$d_{max} = \max_{j\in K}\left\{-\bar{c}_j + \sum_{i\in M}\bar{A}_{ij}y_i^0 + \bar{s}_j\right\}.$$

Applying l_∞ norm to (6.14) and considering the upper bound of (6.15), we obtain

$$\begin{aligned} ||\Delta s||_\infty &\leq ||A^T||_\infty ||\Delta \bar{y}||_\infty. \\ &\leq \frac{\sqrt{m}\ \sigma_{max}(A)\ d_{max}}{\bar{A}_{min}}. \end{aligned}$$

Thus,

$$\Delta s_j \leq \frac{\sqrt{m}\ \sigma_{max}\ d_{max}}{\bar{A}_{min}}, \quad \forall j \in N. \tag{6.16}$$

Additionally, considering (6.12), for every $j \in K$, we have

$$\bar{s}_j + \Delta \bar{s}_j \leq \frac{\sqrt{m}\ \sigma_{max}\ d_{max}}{\bar{A}_{min}} + (\bar{c}_j - \bar{A}_j^T y^0), \quad \forall j \in K. \tag{6.17}$$

6.2.3 Primal feasibility restoration

Similar to the recovery of dual feasibility, we aim to restore feasibility in the primal space by solving an auxiliary linear optimization problem and using the notion of primal dominant and dual dominant partitions. Considering the first equation of system (6.2) and $(x^0, y^0, s^0) \in \mathcal{F}^0$, we have

$$A\Delta x = -\bar{A}(\bar{x} + \Delta \bar{x}) \tag{6.18}$$

To simplify the notation, let us define $\bar{x}_j^w = \bar{x}_j + \Delta \bar{x}_j$ for every $j \in K$. Primal feasibility could be easily restored if we could set $\bar{x}^w = 0$. Since in interior point methods this is not possible, we need to fix or determine a positive value for $\bar{x}^w$. In practice, primal feasibility could still be easily achieved by setting $\bar{x}^w$ sufficiently small. We could apply for example the primal feasibility restoration direction proposed in [56], $\Delta x = -\Theta A^T (A\Theta A^T)^{-1} \bar{A}\bar{x}^w$ for non-degenerate systems where $\Theta = XS^{-1}$ is a diagonal scaling matrix. This is a generalization of Mitchell's direction presented in [93] and applied in the primal projective algorithm to handle multiple cuts. Setting a small value for $\bar{x}^w$ seems to be sensible since primal infeasibility depends on this value. We have designed a slightly different strategy which takes into account setting x_j^0 small but also considers centrality aspects. For now, it is enough to say that our choice of $\bar{x}^w$ is the maximum possible value in order to ensure that: (a) a full step in the primal direction is possible, and (b) primal feasibility is restored. Similar to the dual feasibility restoration, we have defined the following auxiliary linear optimization problem.

$$\begin{aligned}
\mathcal{P}_{aux} := \quad & \text{minimize} && \sum_{j \in \mathcal{N}_0} \frac{\Delta x_j^+ + \Delta x_j^-}{x_j^0}, && && \text{(6.19a)} \\
& \text{subject to} && \sum_{j \in N} A_{ij} \Delta x_j = -\sum_{j \in K} \bar{A}_{ij} \bar{x_j}^w, && \forall i \in M, && \text{(6.19b)} \\
& && \Delta x_j \geq x_j^0 (\delta_l - 1), && \forall j \in N, && \text{(6.19c)} \\
& && \Delta x_j = \Delta x_j^+ - \Delta x_j^-, && \forall j \in \mathcal{N}_0, && \text{(6.19d)} \\
& && \Delta x_j^+ \geq 0, && \forall j \in \mathcal{N}_0, && \text{(6.19e)} \\
& && \Delta x_j^- \geq 0, && \forall j \in \mathcal{N}_0, && \text{(6.19f)}
\end{aligned}$$

where δ_l is a given parameter which satisfies $0 < \delta_l < 1$ and its meaning will be

explained later in this chapter. In the primal case, and similar to what we did for the dual variables in the primal dominant partition, we minimize the relative change of the variables in the dual dominant partition (corresponding to small x_j^0). Note that in this case we allow positive and negative directions and therefore we minimize the absolute value of such directions. Constraints (6.19b) guarantee that by taking a full step in direction Δx, feasibility in the primal space is completely restored. Constraints (6.19c) ensure that if we take a full step in direction Δx, the new iterate will remain positive. Constraints (6.19d)-(6.19f) are additional requirements that help to calculate the absolute value of every Δx_j in the dual dominant partition. If $\Delta x_j > 0$, then $\Delta x_j^+ > 0$ and $\Delta x_j^- = 0$. If $\Delta x_j < 0$, then $\Delta x_j^- > 0$ and $\Delta x_j^+ = 0$. Finally, if $\Delta x_j = 0$, then both Δx_j^+ and Δx_j^- are zero. Note that ensuring a non-empty feasible set in $\mathcal{P}_{aux}$ for general A and $\bar{A}$ is a non-trivial task. For instance, large values on $\bar{x}^w$ may lead to large negative values in some Δx_j and therefore, satisfying constraint (6.19c) may not be possible. Later we will define conditions to ensure that primal infeasibility is completely absorbed in one step.

Following [123], let us use the following QR factorization of A^T,

$$A^T = Q \begin{bmatrix} R \\ 0 \end{bmatrix} = [Q_1, Q_2] \begin{bmatrix} R \\ 0 \end{bmatrix} = Q_1 R, \tag{6.20}$$

where Q_1 is an $n \times m$ matrix, Q_2 is an $n \times (n-m)$ matrix, $[Q_1, Q_2]$ is a matrix with orthogonal columns, and R is an $m \times m$ upper triangular matrix. It is easy to check that for the aforementioned factorization and given $\bar{x}^w$

$$\Delta x \quad = \quad -Q_1 R^{-T} \bar{A} \bar{x}^w, \tag{6.21}$$

satisfies equation (6.18). If $\mathcal{N}_0 = \emptyset$, problem (6.19) becomes a feasibility problem and it only requires to find a solution which solves (6.19b) and (6.19c). Such a solution can be obtained from (6.21) given a suitable value of $\bar{x}_j^w$ for every $j \in K$, as we will show later. It is known as the minimum norm solution. However, this is not the only solution Δx which satisfies (6.18). Applying the l_2 norm to (6.21), we have

$$||\Delta x|| \leq ||Q_1||\ ||R^{-T}||||\bar{A}\bar{x}^w||.$$

Since $||Q_1|| = 1$, $||R^{-T}|| = \sigma_{min}(R)^{-1}$ and $\sigma_{min}(R) = \sigma_{min}(A)$, it follows that

$$\begin{aligned} ||\Delta x|| \quad &\leq \quad \frac{||\bar{A}\bar{x}^w||}{\sigma_{min}(A)}. \\ &\leq \quad \frac{\sqrt{k}\ \sigma_{max}(\bar{A})\ \bar{x}^w_{max}}{\sigma_{min}(A)}. \end{aligned} \tag{6.22}$$

From (6.22), and using our definition of σ_{max}, we can derive loose upper and lower bounds for every Δx_j, with $j \in N$. These bounds are

$$-\frac{\sqrt{k}\ \sigma_{max}\ \bar{x}^w_{max}}{\sigma_{min}(A)} \leq \Delta x_j \leq \frac{\sqrt{k}\ \sigma_{max}\ \bar{x}^w_{max}}{\sigma_{min}(A)}, \tag{6.23}$$

and problem (6.19) has a bounded solution.

Let us define the feasible set of Δx associated with $\mathcal{P}_{aux}$ as $\mathcal{S}_\mathcal{P} = \{\Delta x : (6.19b) - (6.19c) \text{ are satisfied}\}$. In addition, let us define a closely related set $\mathcal{S}_\mathcal{Q} = \{\Delta x : (6.19b) \text{ are satisfied}\}$. Observe that Δx from (6.21) satisfies (6.19b) hence $\Delta x \in \mathcal{S}_\mathcal{Q}$.

However, it does not have to satisfy (6.19c) and therefore in general $\Delta x \notin \mathcal{S}_{\mathcal{P}}$. The next lemma states that given a particular choice of $\bar{x}^w$ and the correct choice of μ_0, we can guarantee that $\Delta x \in \mathcal{S}_{\mathcal{P}}$.

Lemma 6.5. *Let* $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$ *for* $\gamma \in (0,1)$, $A_{ij} \geq 0, \forall i \in M, \forall j \in N$, $\bar{A}_{ij} \geq 0, \forall i \in M, \forall j \in K$. *If*

$$\bar{x}_j^w \leq \gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\ \mathcal{U}_v\ \sigma_{max}} \min\left\{\mu_0, \frac{1}{\mu_0}\right\}, \quad \forall j \in K, \tag{6.24}$$

and $0 < \delta_l < 1$, *then constraint* (6.19c) *is satisfied.*

Proof. To ensure this, we have to show that (6.19c) is satisfied for every Δx_j from (6.23). Hence, we have to prove that

$$\Big(1-\delta_l\Big)x_j^0 \geq \frac{\sqrt{k}\ \sigma_{max}\ \bar{x}_{max}^w}{\sigma_{min}(A)}.$$

Since we need to ensure this for every $j \in N$, it suffices if we check that this inequality is satisfied for the smallest possible x_j^0. We know from (6.6a) that $x_j^0 \geq (\gamma\mu_0)/\mathcal{U}_v, \forall j \in N$. Therefore, we need to verify that

$$\Big(1-\delta_l\Big)\frac{\gamma\mu_0}{\mathcal{U}_v} \geq \frac{\sqrt{k}\ \sigma_{max}\ \bar{x}_{max}^w}{\sigma_{min}(A)}.$$

This condition is satisfied if we choose $\bar{x}_j^w$ for every $j \in K$ satisfying (6.24) and by noting that $\mu_0 \geq \min\{\mu_0, 1/\mu_0\}$. □

Note that with our choice of $\bar{x}_j^w$ satisfying (6.24) for every $j \in K$, we can find a solution to problem (6.19). As in the dual case, we now guarantee that a full step can be taken in direction Δx.

Lemma 6.6. *Let* $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$ *with* $\gamma \in (0,1)$ *and* $0 < \delta_l < 1$. *By choosing* Δx *from* (6.19) *and setting*

$$\bar{x}_j^w = \gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\ \mathcal{U}_v\ \sigma_{max}} \min\left\{\mu_0, \frac{1}{\mu_0}\right\}, \quad \forall j \in K, \tag{6.25}$$

we have $x_j^0 + \Delta x_j \geq 0$ *and a full step in the primal space is feasible. Therefore, primal feasibility is restored in one step.*

Proof. Similarly to Lemma 6.4, from constraints (6.19c) and since $x_j^0 > 0$ and $0 < \delta_l < 1$, we deduce $x_j^0 + \Delta x_j > 0$ as required. Since this solution satisfies condition (6.19b), we conclude that the warmstart is primal feasible. □

Additionally, using the bounds in (6.23), the choice of $\bar{x}^w$ in (6.25) and since $0 < \delta_l < 1$, we can guarantee

$$-(1-\delta_l) \leq \frac{\Delta x_j}{x_j^0} \leq (1-\delta_l). \tag{6.26}$$

Note that our choice of $\bar{x}_j^w$ is independent of $\bar{s}_j^w$ for every $j \in K$. Moreover, and as described in Section 6.2.2, $\bar{s}_j^w$ for every $j \in K$ is not fixed and must be calculated. However, and as we will see in the next section, both values are related via other parameters (*i.e.*, μ_0 and γ) in order to guarantee that complementarity products are

still inside a slightly modified neighbourhood and that the new duality gap is also bounded.

Summarizing our findings so far, we have shown that by the use of $\mathcal{P}_{aux}$ and $\mathcal{D}_{aux}$ and choosing carefully $\bar{s}_j$ and $\bar{x}_j^w$, for every $j \in K$, we could take a full step in direction $(\Delta x, \Delta y, \Delta s)$ recovering primal and dual feasibilities. Now, we will analyse the consequences that such warmstarting approach has in the complementarity products.

6.2.4 Centrality

The main motivation why we have chosen to minimize a variation of the weighted-least squares approach in our linear optimization problems is to avoid having large variations $(\Delta x, \Delta s)$ on small components. We would like to keep the terms $|\Delta x_j|/|x_j^0|$, $j \in \mathcal{N}_0$ and $|\Delta s_j|/|s_j^0|$, $j \in \mathcal{B}_0$ bounded by some constant so that we could have some control on the complementarity products (centrality) of the warmstarting point and therefore, a control on the new duality gap.

To analyse the effect of our warmstarting strategy on centrality of the new iterate, let us first determine the depth of the cut for which our analysis holds. As stated before, the depth of cut j is defined by $-(\bar{c}_j - \bar{A}_j^T y^0)$ for every $j \in K$. It is not surprising that if the depth of the cuts is large, we will need to backtrack to an earlier iterate. This is considered in our analysis by the notion of μ_0 which is a reference barrier term that measures the depth of the cuts. When cuts are deep, μ_0 is large so we may need to backtrack far from optimality, but when the cuts get shallower, μ_0 gets smaller and therefore, we could choose an iterate close to optimality. Now, let us state the relation between the depth of the cuts and μ_0 using the following expression for every $j \in K$

$$\underbrace{\frac{1}{\gamma}\mu_0 \frac{\sqrt{k}\,\mathcal{U}_v\,\sigma_{max}}{(1-\delta_l)\,\sigma_{min}(A)} \max\left\{\mu_0, \frac{1}{\mu_0}\right\}}_{\mathcal{U}_c} \geq \bar{c}_j - \bar{A}_j{}^T y^0 \geq -\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}. \tag{6.27}$$

Note that we have some control with regards to the size of the cut. This means that we could retreat further back in the list of saved iterates to find a suitable large enough μ_0 and the corresponding (x^0, y^0, s^0) solution for which this condition is satisfied. Observe that increasing μ_0 will expand both sides of inequality (6.27) increasing the chances to satisfy this condition. Also, observe that the left hand side inequality provides us with a very loose upper bound when, by retreating back in the list of stored iterates, some of the reduced costs become positive. This bound becomes useful when defining the upper bound of the complementarity products obtained by our warmstarting strategy. Finally, it is fair to say that in some iterations (6.27) may not be satisfied by any of the stored iterates and therefore, we use coldstart instead.

Now, using definition (6.27) and the definition of $\bar{s}_j$ in (6.10), from (6.12) we deduce for every $j \in K$ that

$$\begin{aligned} d_j &= -(\bar{c}_j - \bar{A}_j^T y^0) + \bar{s}_j, \\ &\leq 2\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}. \end{aligned}$$

Hence,

$$d_{max} \quad = \quad 2\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}. \tag{6.28}$$

Substituting d_{max} in (6.16) and (6.17) we obtain

$$\Delta s_j \quad \leq \quad 2\gamma\mu_0 \frac{1}{\mathcal{U}_v}, \quad \forall j \in N, \tag{6.29}$$

and

$$\bar{s}_j + \Delta\bar{s}_j \quad \leq \quad 2\gamma\mu_0 \frac{1}{\mathcal{U}_v} + \mathcal{U}_c, \quad \forall j \in K, \tag{6.30}$$

respectively, where $\mathcal{U}_c$ denotes the left hand side term of inequality (6.27).

Now, let us analyse the consequences of our choices of $\bar{s}^w$ and $\bar{x}^w$ and their impact on the complementarity products of the warmstarting iterate. To do so, first let us define δ_l as

$$\delta_l := \min\left\{\frac{\mathcal{C}}{\mu_0 + \mathcal{C}}, \frac{\mathcal{C}}{\frac{1}{\mu_0} + \mathcal{C}}\right\}, \tag{6.31}$$

where

$$\mathcal{C} := \frac{\gamma\sigma_{min}(A)\bar{A}_{min}}{\sqrt{k}\,\sqrt{m}\,\mathcal{U}_v^2\,\sigma_{max}^2}. \tag{6.32}$$

From (6.31) it is clear that $0 < \delta_l < 1$. Additionally, let us define δ_u as

$$\delta_u := 6. \tag{6.33}$$

These values represent the coefficients used to expand the neighbourhood from below and above, respectively. As mentioned earlier, we expect the new warmstarting point to be inside a modified neighbourhood. Note that the parameter δ_l contains valuable information regarding the old problem and the new columns appended to it, through matrices A and $\bar{A}$. Also, note that δ_u is problem independent. As we will discuss later, this is one of the key results of the analysis since it guarantees that regardless the size of the cut, and if some conditions hold, the new duality gap will be bounded by the old duality gap and a constant.

The following theorem states conditions and analyses complementarity products when our specialized warmstarting strategy is used. Let us introduce the following notation: $\hat{N} = \{1, \ldots, n, n+1, \ldots, n+k\}$, $\hat{x}^w = [x^w; \bar{x}^w]$, $\hat{s}^w = [s^w; \bar{s}^w]$, $\hat{c} = [c; \bar{c}]$ and $\hat{A} = \begin{bmatrix} A & \bar{A} \end{bmatrix}$.

Theorem 6.7. *Let assumptions (A.1)-(A.4) hold and let $(x^0, y^0, s^0) \in \mathcal{N}_s(\gamma)$ with $\gamma \in (0, 1)$. Additionally, let δ_l and δ_u be defined by* (6.31) *and* (6.33)*, respectively. Also, let us set*

$$\bar{x}_j^w = \gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\,\mathcal{U}_v\,\sigma_{max}} \min\left\{\mu_0, \frac{1}{\mu_0}\right\}, \quad \forall j \in K,$$

and

$$\bar{s}_j = \max\left\{\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}, \bar{c}_j - \bar{A}_j^T y^0\right\}, \quad \forall j \in K.$$

Having $\bar{c}_j - \bar{A}_j{}^T y^0$ satisfying (6.27)*, for every $j \in K$, and choosing Δx from* (6.19)

and $(\Delta y, \Delta s)$ *from* (6.11) *where* $\Delta y = -\Delta\bar{y}$*, we have that*

$$(\hat{x}^w, y^w, \hat{s}^w) \in \hat{\mathcal{N}}_s(\gamma),$$

where

$$\hat{\mathcal{N}}_s(\gamma) = \left\{(\hat{x}^w, y^w, \hat{s}^w) \in \hat{\mathcal{F}}^0 : \delta_l\gamma\mu_0 \leq \hat{x}_j^w\hat{s}_j^w \leq \delta_u\frac{1}{\gamma}\mu_0, \forall j \in \hat{N}\right\},$$

and

$$\hat{\mathcal{F}}^0 = \left\{(\hat{x}^w, y^w, \hat{s}^w) : \hat{A}\hat{x}^w = b; \hat{A}^T y^w + \hat{s}^w = \hat{c}; (\hat{x}^w, \hat{s}^w) > 0\right\}.$$

Proof. It is not difficult to show that by choosing δ_l from (6.31) and δ_u from (6.33), and using our previous choices for $\bar{x}_j^w$ (see (6.25)), and $\bar{s}_j^w = \bar{s}_j + \Delta\bar{s}_j$ (see (6.10) and (6.30)), the complementarity products for the new components, namely $\bar{x}_j^w\bar{s}_j^w$ for every $j \in K$ which correspond to $\hat{x}_j^w\hat{s}_j^w$ for every $j \in \{n+1, \ldots, n+k\}$, are inside the modified neighbourhood, $\hat{\mathcal{N}}_s(\gamma)$. Let us prove first that the upper bound holds. For every $j \in K$, we have that

$$\begin{aligned}\bar{x}_j^w\bar{s}_j^w &= \bar{x}_j^w(\bar{s}_j + \Delta\bar{s}_j), \\ &\leq \gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\,\mathcal{U}_v\,\sigma_{max}}\min\left\{\mu_0, \frac{1}{\mu_0}\right\}\left(2\gamma\mu_0\frac{1}{\mathcal{U}_v} + \mathcal{U}_c\right).\end{aligned}$$

Since $0 < \gamma < 1$, $0 < \delta_l < 1$, $\mathcal{U}_v^2 > 1$, $k \geq 1$, $\min\{\mu_0, 1/\mu_0\} \leq 1$, and the definition of $\mathcal{U}_c$, it is clear that

$$\begin{aligned}\bar{x}_j^w\bar{s}_j^w &\leq 2\gamma\mu_0\frac{1}{\mathcal{U}_v}\gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\,\mathcal{U}_v\,\sigma_{max}}\min\left\{\mu_0, \frac{1}{\mu_0}\right\} + \mu_0, \\ &< \delta_u\mu_0, \quad \forall j \in K,\end{aligned} \tag{6.34}$$

holds. If we now consider the lower bound for the complementarity products of the new components, we have that for every $j \in K$

$$\begin{aligned}\bar{x}_j^w\bar{s}_j^w &= \bar{x}_j^w(\bar{s}_j + \Delta\bar{s}_j), \\ &\geq \gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\,\mathcal{U}_v\,\sigma_{max}}\min\left\{\mu_0, \frac{1}{\mu_0}\right\}\left(\gamma\mu_0\frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\sigma_{max}}\right).\end{aligned}$$

Using (6.32), we get

$$\bar{x}_j^w\bar{s}_j^w \geq \gamma\mathcal{C}(1-\delta_l)\mu_0\min\left\{\mu_0, \frac{1}{\mu_0}\right\}, \quad \forall j \in K.$$

Then, by choosing δ_l from (6.31), we have

$$\bar{x}_j^w\bar{s}_j^w \geq \delta_l\gamma\mu_0, \quad \forall j \in K, \tag{6.35}$$

Hence, and considering $\gamma \in (0, 1)$, (6.34) and (6.35) the following result holds

$$\delta_l\gamma\mu_0 \leq \bar{x}_j^w\bar{s}_j^w \leq \delta_u\frac{1}{\gamma}\mu_0, \quad \forall j \in K, \tag{6.36}$$

which completes the first part of the prove.

Now, we prove that the complementarity products of the old components, namely

$x_j^w s_j^w$ for every $j \in N$, corresponding to $\hat{x}_j^w \hat{s}_j^w$ for every $j \in \{1, \ldots, n\}$, are inside the modified neighbourhood, $\hat{\mathcal{N}}_s(\gamma)$. Firstly, let us prove that the upper bound holds for every $j \in N$. By definition and conditions (6.6) and (6.26), we have that for every $j \in N$

$$\begin{aligned} x_j^w s_j^w &= x_j^0 s_j^0 \Big(1 + \frac{\Delta x_j}{x_j^0}\Big)\Big(1 + \frac{\Delta s_j}{s_j^0}\Big), \\ &\leq x_j^0 s_j^0 \Big(2\Big)\Big(1 + \Delta s_j \frac{\mathcal{U}_v}{\gamma \mu_0}\Big). \end{aligned}$$

Hence, and by using (6.29) and (6.33), we get

$$x_j^w s_j^w \leq \delta_u \frac{1}{\gamma} \mu_0, \quad \forall j \in N.$$

To prove that the lower bound holds we recall constraints (6.19c) which guarantee

$$\Delta x_j \geq x_j^0 (\delta_l - 1), \quad \forall j \in N. \tag{6.37}$$

Also, from (6.9), $\Delta s_j \geq 0$. Hence for every $j \in N$

$$\begin{aligned} x_j^w s_j^w &= x_j^0 s_j^0 \Big(1 + \frac{\Delta x_j}{x_j^0}\Big)\Big(1 + \frac{\Delta s_j}{s_j^0}\Big), \\ &\geq x_j^0 s_j^0 \Big(1 + \frac{\Delta x_j}{x_j^0}\Big). \end{aligned} \tag{6.38}$$

From (6.37) and since $x_j^0 > 0$, we know that

$$1 + \frac{\Delta x_j}{x_j^0} \geq \delta_l, \quad \forall j \in N.$$

Replacing this in (6.38) gives

$$x_j^w s_j^w \geq \delta_l x_j^0 s_j^0, \quad \forall j \in N,$$

which completes the proof. □

6.2.5 Algorithm

Now we are in position to describe the algorithm proposed to find a warmstarting point after columns with reduced costs satisfying $\bar{c}_j - \bar{A}_j{}^T y < 0$ are added to the RMP. Note that this algorithm is embedded inside a major algorithm which is the primal-dual column generation method (see Algorithm 5.1). Let us define $T = \{1, 2, \ldots, h\}$ as the set of indices of iterates which are strictly feasible and well-centred in the initial problem. Observe that T, h and their corresponding list of stored solutions vary at each outer iteration of PDCGM. The list is created in ascending order so the last stored iterate and the closest-to-optimality solution is denoted by (x^h, y^h, s^h). Algorithm 6.1 summarizes the principal steps of our specialized warmstarting strategy.

Algorithm 6.1 One Step Primal-Dual Warmstarting Strategy

Input: A, b, c, $\bar{c}$, $\bar{A}$, $\mathcal{U}_v$, n, k, m, $\gamma \in (0,1)$, list of $(x^t, y^t, s^t) \in \mathcal{N}_s(\gamma)$ and $\mu_t = (x^t)^T s^t/n$, where $t \in T$.

Steps:

1: If no column is returned by the oracle, set $(x^w, y^w, s^w) = (x^h, y^h, s^h)$, and then go to 8. Otherwise, go to 2.

2: Calculate the smallest μ from the list of stored iterates such that

$$\mathcal{U}_c \geq \bar{c}_j - \bar{A}_j{}^T y^0 \geq -\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}},$$

is satisfied. If there exists such μ, define $\mu = \mu_0$, denote its associated solution as (x^0, y^0, s^0), and go to 3. If not, use coldstart and go to 8.

3: Define

$$\delta_l := \min\{\frac{\mathcal{C}}{\mu_0 + \mathcal{C}}, \frac{\mathcal{C}}{\frac{1}{\mu_0} + \mathcal{C}}\}, \quad \text{where } \mathcal{C} := \frac{\gamma \bar{A}_{min}\sigma_{min}(A)}{\sqrt{k}\,\sqrt{m}\,\mathcal{U}_v^2\,\sigma_{max}^2}.$$

4: For every $j \in K$, set

$$\bar{s}_j = \max\left\{\gamma\mu_0 \frac{\bar{A}_{min}}{\sqrt{m}\,\mathcal{U}_v\,\sigma_{max}}, \bar{c}_j - \bar{A}_j^T y^0\right\},$$

$$\bar{x}_j^w = \gamma(1-\delta_l)\frac{\sigma_{min}(A)}{\sqrt{k}\,\mathcal{U}_v\,\sigma_{max}} \min\left\{\mu_0, \frac{1}{\mu_0}\right\},$$

and define $\mathcal{B}_0$ and $\mathcal{N}_0$.

5: Solve $\mathcal{D}_{aux}$. Output: $(\Delta y, \Delta s)$.

6: Solve $\mathcal{P}_{aux}$. Output: Δx.

7: Define $(x^w, \bar{x}^w, y^w, s^w, \bar{s}^w) = (x^0 + \Delta x, \bar{x}^w, y^0 + \Delta y, s^0 + \Delta s, \bar{s}^w)$.

8: Continue with the usual primal-dual interior point method by solving the Newton system of equations in the old/new dimensions.

Observe that a new list of stored iterates is used every time we initialize Algorithm 6.1. Note that Step 1 in Algorithm 6.1 accounts for the case in which no columns with negative reduced costs are obtained from the oracle. This is a particular feature of the PDCGM (as explained in Chapter 5). There is no point in using a warmstarting strategy since the current iterate is already well-centred and strictly feasible and therefore starting from it is the best we could do. Also, it is important to note that every cut $\bar{c}_j - \bar{A}_j{}^T y^h < 0$ is generated using (x^h, y^h, s^h). Hence, for any iterate (x^t, y^t, s^t), where $t \in T$, some of the new columns (cuts) may actually become dual feasible, or at least less infeasible (shallower cuts). The following theorem summarizes the main discussion and results of this section.

Theorem 6.8. *Using Algorithm 6.1, and given that a suitable (x^0, y^0, s^0) is available from the list of iterates, the solution $(\hat{x}^w, y^w, \hat{s}^w) \in \hat{\mathcal{N}}_s(\gamma)$ and the new duality gap is bounded by*

$$(\hat{x}^w)^T \hat{s}^w \leq 6\big(n+k\big)\frac{1}{\gamma}\mu_0. \tag{6.39}$$

Proof. The proof follows from Lemmas 6.4 and 6.6 and Theorem 6.7. □

In summary, our strategy aims to restore primal and dual feasibilities separately by means of auxiliary linear optimization problems. We can ensure that primal and

dual feasibilities are fully recovered if a suitable μ can be found. Note that Theorem 6.8 provides a guarantee that the new duality gap is bounded by the old duality gap multiplied by a constant which depends on the number of columns added to the old problem. Our analysis takes into account special classes of problems arising in combinatorial optimization with non-negative technological coefficients.

6.3 Computational study

We have implemented the proposed warmstarting strategy inside the primal-dual column generation method. This column generation implementation has a built-in procedure which allows to store an advanced iterate in order to use it for future warmstart. We have taken advantage of this to implement our strategy. We have tested three different strategies to solve the relaxation of the one-dimensional cutting stock problem (CSP) and the vehicle routing problem with time windows (VRPTW) after applying the Dantzig-Wolfe reformulation described in Chapter 4.

The strategies considered to find a suitable candidate from which to start solving each RMP are:

- Coldstart (CS). It refers to solving a given RMP without considering any information of the previously solved RMP and therefore, every RMP is solved from scratch. We rely on the presolving and heuristic procedures included in HOPDM [54].

- Partial feasibility restoration (PFR). This strategy was introduced in [56] and has been successfully applied to speed up the column generation procedure in [62]. This strategy ensures that dual feasibility is fully restored in the new components if cuts are not too deep. However, it does not ensure the same in the old components. Additionally, it sets the new components in the primal space to a small value so primal feasibility can be easily restored. Moreover, this strategy is the one used in the computational experiments presented in Chapter 5.

- One step primal-dual restoration (1SPDR). This strategy is described in Algorithm 6.1 and aims to recover primal and dual feasibilities in one step by solving two auxiliary problems. Also, theoretical guarantees are provided so the warmstarting iterate stays close to the new central path.

Similar to the previous computational experiments, the subproblems obtained after applying the reformulation to CSP and VRPTW are solved using the same source code, *i.e.,* knapsack solver [82] (CSP) and our own implementation of the bounded bidirectional dynamic programming algorithm proposed in [107], with state-space relaxation and identification of unreachable nodes [37] (VRPTW). For the three aforementioned strategies (CS, PFR and 1SPDR), the RMPs are initialized with the same columns. However, after the first iteration every RMP may be different and therefore, we may expect that some of the strategies will require fewer outer iterations than others. To run the tests we have used a laptop with a 2.3 Ghz Intel Core i5 processor, 4 GB RAM and a Linux operating system. The implementations have been developed in C (solving the two auxiliary problems) and FORTRAN (HOPDM native language). For each of the strategies, we stop the column generation procedure when the relative duality gap becomes smaller than the default accuracy $\delta = 10^{-6}$. Note that the results may be different from the ones presented in the previous chapter. This is due to using a different machine to run the experiments.

In the results presented in Chapter 5, it was observed that the time spent solving the RMP was considerable when compared with the time spent in the oracle when solving CSP. On the other hand, the dominant task when solving VRPTW was the oracle and the RMP time only accounted for a small portion of the total time. By presenting the results for these two applications, we could get a better understanding of the situations where we can expect a warmstarting strategy to perform better than coldstart and how much the gain can be.

Before continuing, it is important to give some remarks about our implementation. Note that for advanced column generation iterations, the coefficients of the objective function, in the primal and dual auxiliary problems ($1/s_j^0$ in (6.9) and $1/x_j^0$ in (6.19)), can lead to very badly-scaled problems (some very large coefficients and others very small). Therefore, we have decided to scale these coefficients and restrict them to a narrower range. In other words, if a given coefficient is greater/smaller than a predefined threshold, we have used this threshold as the coefficient for this specific variable. Note that by doing this, we do not affect the primal and dual feasible sets, only the scaling of that particular auxiliary problem and therefore, the strategy still recovers primal and dual feasibility in one step. Additionally, since PDCGM is based on HOPDM [54], an infeasible primal-dual interior point method is used to solve every RMP which keeps the iterates inside a neighbourhood of the central path by making use of multiple centrality corrector steps [21]. We have considered this in our developments in the following way. When the column generation process approaches optimality and the depth of the cuts is small, that is when μ and $(\bar{c}_j - \bar{A}_j^T y_j^0)$ are small, instead of solving the primal and dual auxiliary problems, we set the values of new components to $\bar{x}_j^w = \sqrt{\mu/\gamma}$ and $\bar{s}_j^w = \sqrt{\gamma\mu}$ for every $j \in K$. We keep the old iterates unchanged. This choice is justified since the cuts at this stage are likely to be shallow and therefore, by choosing the new components in this way, we only generate small infeasibilities which HOPDM (and any infeasible primal-dual interior point method) can easily handle. Additionally, with these choices, we ensure that the complementarity products for the new iterates are inside the neighbourhood described in (6.5). Moreover, and after some testing, the directions, and therefore, the correction steps, obtained by solving the auxiliary problems in an advanced stage of the column generation process, are very short and do not change significantly the stored iterate. By performing this small change in the algorithm, we aim to reduce the CPU time of the method by avoiding unnecessary calls to the auxiliary problems and taking advantage of the infeasible primal-dual interior point method. Finally, to solve the primal and dual auxiliary problems we rely on the solver HOPDM.

6.3.1 Cutting stock problem

To analyse the performance of these three strategies for solving CSP, we have selected 262 instances from the literature in the one-dimensional CSP (`http://www.math.tu-dresden.de/~capad/`). The size of the instances vary between 15 and 285 items. The column generation procedure is initialized with columns generated by homogeneous cutting patterns, which corresponds to selecting only one piece per pattern, as many times as possible without violating the width of the rolls.

In Table 6.1, we summarize our findings. In the first column we denote by k the number of columns added at each iteration of the column generation procedure. Note that by adding more columns at every iteration we are targeting to reduce the number of calls to the oracle. However, in terms of warmstarting this may have an important impact since the similarities between two consecutive RMPs are likely to be lost. We

have grouped the instances in the same way as in Chapter 5 (for number of instances per class, refer to Table 5.1). In row ALL, we have included the average results when all the instances are considered for that particular value of k. For each strategy we show the average number of inner iterations (inner) and the average CPU time required to solve the RMPs (rmp(s)). Inner iterations refer to the total number of iterations required to solve the RMPs while the RMP time considers the time required to solve the RMPs and the time of any warmstating procedure. For instance, for 1SPDR, it is the overall time of solving the primal and dual auxiliary problems and the time of solving each RMP. Moreover, we also include the total time required for the PDCGM to converge to an optimal solution (tot(s)).

Table 6.1: Average results on 262 instances of CSP for PFR, CS and 1SPDR strategies adding k columns at a time: RMP iterations and times (RMP and total).

		PFR			**CS**			**1SPDR**			**PFR/1SPDR**		**CS/1SPDR**	
k	**class**	**inner**	**rmp(s)**	**tot(s)**	**inner**	**rmp(s)**	**tot(s)**	**inner**	**rmp(s)**	**tot(s)**	**inner**	**tot(s)**	**inner**	**tot(s)**
1	U	2628.2	26.3	144.0	7714.9	100.9	257.1	5953.8	52.2	167.9	0.4	0.9	1.3	1.5
	mX	3171.4	13.5	23.7	5973.3	28.2	40.1	2920.4	10.1	18.8	1.1	1.3	2.0	2.1
	MTP0xJES	927.7	2.5	4.5	2879.3	4.6	6.3	1752.3	3.1	4.6	0.5	1.0	1.6	1.3
	MTP0	959.2	2.7	4.3	3034.8	5.1	6.8	1762.6	3.6	5.0	0.5	0.8	1.7	1.3
	hard28	1895.5	6.6	8.1	5210.6	14.1	15.4	2396.8	6.6	7.7	0.8	1.1	2.2	2.0
	7hard	1279.0	3.7	4.5	3351.9	6.7	7.4	1814.0	4.1	4.6	0.7	1.0	1.8	1.6
	53NIRUPs	1063.9	2.8	3.4	2755.6	5.2	5.8	1448.4	2.9	3.4	0.7	1.0	1.9	1.7
	gau3	488.0	0.6	0.6	692.0	0.7	0.7	321.0	0.4	0.4	1.5	1.6	2.2	1.9
	ALL	**2438.5**	**10.9**	**25.9**	**5192.1**	**26.2**	**45.0**	**2723.2**	**11.1**	**25.0**	**0.9**	**1.0**	**1.9**	**1.8**
10	U	765.2	6.6	52.5	2721.7	36.9	88.3	1443.5	14.8	49.2	0.5	1.1	1.9	1.8
	mX	1035.5	6.1	10.4	1992.5	10.5	15.0	1073.9	5.6	9.8	1.0	1.1	1.9	1.5
	MTP0xJES	278.0	0.8	1.5	741.0	1.4	2.2	486.0	1.1	1.9	0.6	0.8	1.5	1.1
	MTP0	307.8	0.8	1.6	771.6	1.7	2.3	467.0	1.2	1.9	0.7	0.8	1.7	1.2
	hard28	654.5	2.9	3.8	1739.0	6.8	7.7	911.3	3.6	4.4	0.7	0.9	1.9	1.7
	7hard	379.0	1.2	1.5	961.0	2.7	3.1	555.0	1.5	1.8	0.7	0.8	1.7	1.7
	53NIRUPs	339.2	1.0	1.4	816.1	2.1	2.4	469.5	1.3	1.5	0.7	0.9	1.7	1.6
	gau3	114.0	0.2	0.2	237.0	0.3	0.3	156.0	0.2	0.2	0.7	1.0	1.5	1.3
	ALL	**789.7**	**4.4**	**10.6**	**1711.2**	**9.9**	**16.5**	**926.8**	**5.0**	**10.1**	**0.9**	**1.0**	**1.8**	**1.6**
50	U	534.3	7.3	59.4	1640.0	24.8	62.5	877.1	12.5	41.3	0.6	1.4	1.9	1.5
	mX	729.7	8.7	19.6	1526.8	17.2	28.4	1090.3	11.8	20.7	0.7	0.9	1.4	1.4
	MTP0xJES	205.7	0.7	2.1	457.3	1.5	2.8	307.3	0.9	2.1	0.7	1.0	1.5	1.4
	MTP0	191.8	0.8	2.3	455.2	1.5	2.7	319.0	1.1	2.3	0.6	1.0	1.4	1.2
	hard28	483.0	4.3	7.6	1200.6	8.5	11.4	686.8	4.7	7.1	0.7	1.1	1.7	1.6
	7hard	277.6	1.4	2.7	669.3	3.1	4.2	413.6	1.7	2.7	0.7	1.0	1.6	1.6
	53NIRUPs	240.4	1.3	2.4	545.3	2.4	3.3	334.2	1.4	2.2	0.7	1.1	1.6	1.5
	gau3	136.0	0.3	0.4	180.0	0.4	0.5	122.0	0.2	0.3	1.1	1.5	1.5	1.5
	ALL	**558.8**	**6.2**	**16.8**	**1241.3**	**13.0**	**22.6**	**832.5**	**8.3**	**15.9**	**0.7**	**1.1**	**1.5**	**1.4**
100	U	493.7	7.9	64.2	1277.6	22.0	59.3	758.3	13.3	47.0	0.7	1.4	1.7	1.3
	mX	688.9	9.1	29.2	1479.3	26.7	52.8	1100.4	21.1	43.7	0.6	0.7	1.3	1.2
	MTP0xJES	181.0	0.6	2.7	371.3	1.4	3.7	274.3	1.2	3.5	0.7	0.8	1.4	1.1
	MTP0	178.8	0.7	2.7	425.4	1.7	4.2	261.6	1.2	3.5	0.7	0.8	1.6	1.2
	hard28	443.9	3.7	9.0	1081.0	9.9	15.8	654.1	6.1	11.4	0.7	0.8	1.7	1.4
	7hard	259.1	1.3	3.5	678.3	4.6	7.1	357.9	2.0	4.1	0.7	0.8	1.9	1.7
	53NIRUPs	212.1	1.1	2.7	495.9	2.7	4.6	292.4	1.6	3.3	0.7	0.8	1.7	1.4
	gau3	103.0	0.2	0.4	206.0	0.4	0.6	100.0	0.2	0.4	1.0	1.0	2.1	1.5
	ALL	**522.1**	**6.3**	**22.8**	**1163.3**	**18.2**	**36.7**	**814.0**	**13.8**	**29.9**	**0.6**	**0.8**	**1.4**	**1.2**

Note that the best performance in terms of CPU time for each of the strategies and each group of instances is obtained when 10 columns are added at each iteration. From our results it seems that by adding 10 columns, the number of column generation iterations is reduced considerably when compared to a single-column approach and at

the same time, each strategy (re)initializes and solves the RMPs efficiently. Note that considering all instances, the best performance is achieved by 1SPDR when adding 10 columns. It is also fair to say that the performance of PFR is comparable to the one proposed in this study when this number of columns is added per iteration. Both warmstarting strategies outperform a coldstart approach in both number of inner iterations and CPU times.

Let us now compare CS and 1SPDR. Since 1SPDR successfully restores primal and dual feasibility in almost all the column generation iterations and keeps the complementarity products of the warmstarting iterate inside a slightly modified neighbourhood, the number of inner iterations to solve the new problem is reduced when compared to CS. The reductions vary between 23% (class U when $k = 1$) and 54% (class hard28 when $k = 1$). Considering all the instances, the reductions in RMPs iterations are between 30% ($k = 100$) and 48% ($k = 1$). The savings in time are due to these reductions and also the efficiency of calculating the new warmstarting iterate. The gains in total CPU time by using 1SPDR range between 19% to 45%.

The differences between 1SPDR and PFR are due to the nature of each of these strategies and the environment for which they were developed. It is necessary to remark that 1SPDR aims to recover primal and dual feasibility while keeping the warmstarted iterate inside a slightly modified neighbourhood, but PFR aims to recover only feasibility in the new dual components and does not ensure the same for the old components. Additionally, the latter was designed to take full advantage of an infeasible primal-dual interior point method while the former has being developed considering a feasible primal-dual interior point method and only takes advantage of the infeasibility nature of the solver at the end of the column generation process, namely when the primal-dual column generation method is close to termination. In general, PFR outperforms 1SPDR in number of inner iterations. This may be due to the nature of PFR, which delivers close to feasible solutions but at the same time, the iterates are far enough from the boundaries. Then, only few centring steps are needed to recover feasibility and return to the neighbourhood of the new central path. On the other hand, 1SPDR ensures primal and dual feasibility restorations. However, and as a consequence of such restorations, it slightly increases the neighbourhood and therefore, at some iterations the warmstarting iterate may be too close to the boundaries. This originates a sometimes excessive number of centring steps in order to return to the neighbourhood of the new central path and therefore has a big impact on the number of iterations. It is important to point out that this belief is just empirical since no theoretical support is given for PFR with respect to complementarity after the restoration is performed. In terms of CPU time, it is not clear which method performs better. However, the best performance in terms of total CPU time is obtained when using 1SPDR when 10 columns are added.

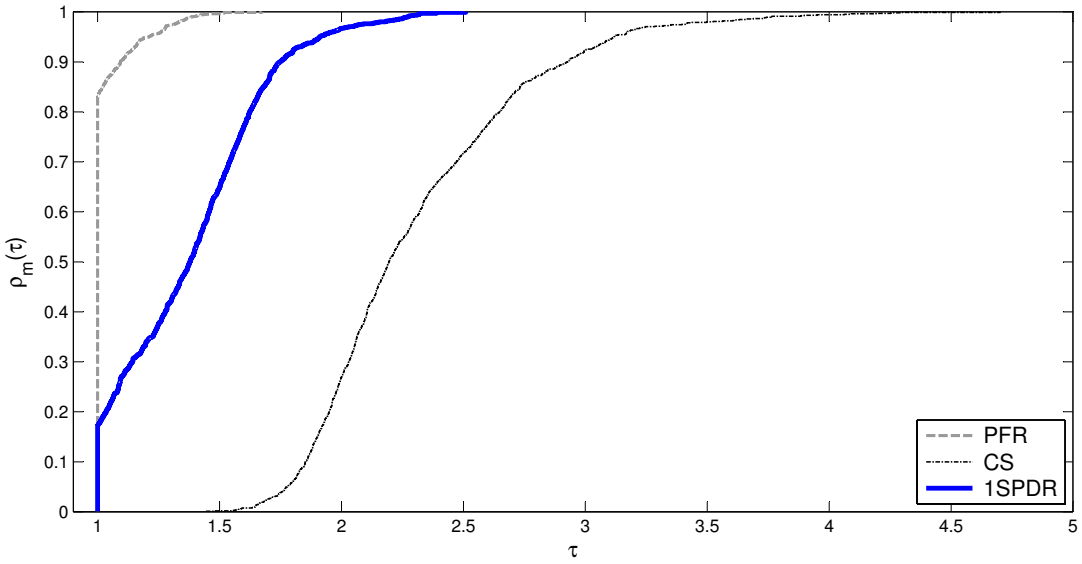
Table 6.2: Average results on 262 instances of CSP for PFR, CS and 1SPDR adding k columns at a time: column generation calls, time and inner iterations per RMP.

	PFR			**CS**			**1SPDR**		
k	**ite**	**rmp(s)/ite**	**inner/ite**	**ite**	**rmp(s)/ite**	**inner/ite**	**ite**	**rmp(s)/ite**	**inner/ite**
1	440.4	0.02	5.5	487.5	0.05	10.7	423.2	0.03	6.4
10	120.0	0.04	6.6	146.2	0.07	11.7	111.9	0.04	8.3
50	73.8	0.08	7.6	91.9	0.14	13.5	67.5	0.12	12.3
100	62.8	0.10	8.3	78.6	0.23	14.8	59.9	0.23	13.6

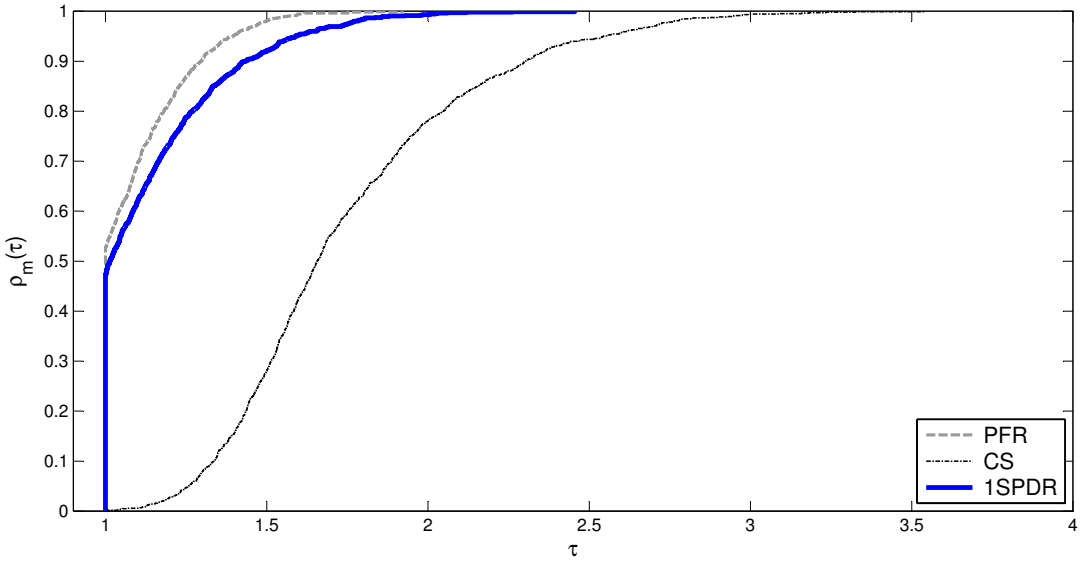
Additionally, in Table 6.2 we present more results of these experiments considering all the instances for different values of k. We have included the number of column generation iterations (ite), the average CPU time required to solve every RMP (rmp(s)/ite) and the average number of inner iterations per RMP (inner/ite).

It is clear from Table 6.2 that by aiming to add more columns to the RMP at each iteration (k), the number of outer iterations is reduced for all the strategies. Also, from Table 6.2 similar behaviour in terms of outer iterations can be observed by every reinitialization strategy for every number of column strategy considered. This result is not surprising since we are using the same column generation strategy and therefore one may expect differences of only few iterations. However, the CPU time is not affected in the same way. This is explained because while adding more columns at each iteration reduces the number of calls to the oracle (which is the most time consuming task for this application), it makes the task of warmstarting more difficult since the new RMP is likely to be very different from the old RMP. This can be seen in the average time required to solve the RMPs. Finally, it seems that 1SPDR is more sensible to the number of columns added per iteration than PFR in terms of inner iterations required to solve the RMPs. Considerable savings in inner iterations are achieved when k is small. However, these benefits start to vanish for large values of k when compared to CS.

Finally, in Figures 6.3(a) and 6.3(b) we have plotted the performance profiles of the PDCGM using these three reinitialization strategies with respect to inner iterations and CPU time, respectively. We have included the results of all the instances and all the values of k considered previously. Note that in terms of inner iterations, PFR is the strategy that requires fewer inner iterations in nearly 80% of the cases while 1SPDR is the best strategy for the remaining 20% for the same performance measure. If we now consider the total CPU time metric, 1SPDR is the best strategy closely followed by PFR. Although PFR requires in general fewer inner iterations, 1SPDR offers reductions in the number of outer iterations and therefore, it performs slightly better. It is clear that CS approach does not offer any benefits for this application in terms of inner iterations nor total CPU time and it is outperformed by one of the warmstarting strategies in all the instances for any value of k.



(a) Inner iterations



(b) Total CPU time

Figure 6.3: Performance profiles for CSP with PFR, CS and 1SPDR

6.3.2 Vehicle routing problem with time windows

From VRPTW literature we have selected 87 instances (`http://www2.imm.dtu.dk/~jla/solomon.html`) originally proposed in [112]. We have used the same classification as in Section 5.2.2. The column generation procedure is initialized with n single-customer routes, which corresponds to assigning one vehicle per customer.

Table 6.3: Average results on 87 instances of VRPTW for PFR, CS and 1SPDR adding k columns at a time: RMP iterations and times (RMP and total).

		PFR			CS			1SPDR			PFR/1SPDR		CS/1SPDR	
k	**class**	**inner**	**rmp(s)**	**tot(s)**	**inner**	**rmp(s)**	**tot(s)**	**inner**	**rmp(s)**	**tot(s)**	**inner**	**tot(s)**	**inner**	**tot(s)**
1	C25	166.2	0.2	0.7	194.3	0.2	0.6	319.1	0.3	0.6	0.5	1.0	0.6	0.9
	R25	206.8	0.3	0.8	203.5	0.2	0.5	371.7	0.3	0.6	0.6	1.3	0.5	0.8
	RC25	227.0	0.4	1.7	281.3	0.2	1.2	412.3	0.4	1.3	0.6	1.3	0.7	0.9
	C50	272.7	0.4	4.4	322.7	0.3	4.3	569.8	0.5	3.7	0.5	1.2	0.6	1.2
	R50	481.9	0.8	6.6	663.0	0.7	5.2	1060.3	1.1	5.3	0.5	1.3	0.6	1.0
	RC50	411.8	0.6	9.4	547.8	0.5	10.0	910.3	0.9	9.3	0.5	1.0	0.6	1.1
	C100	509.6	0.8	39.1	722.8	0.8	40.3	1150.3	1.6	42.0	0.4	0.9	0.6	1.0
	R100	944.5	1.5	139.3	1364.1	1.7	147.4	2574.6	3.8	162.6	0.4	0.9	0.5	0.9
	RC100	797.5	1.2	151.0	1164.1	1.4	155.1	2233.9	2.9	159.2	0.4	0.9	0.5	1.0
	ALL	**455.4**	**0.7**	**39.7**	**619.2**	**0.7**	**41.1**	**1090.6**	**1.4**	**43.7**	**0.4**	**0.9**	**0.6**	**0.9**
10	C25	118.9	0.2	0.4	104.9	0.1	0.3	161.9	0.2	0.3	0.7	1.1	0.6	0.8
	R25	132.8	0.2	0.3	113.4	0.1	0.3	175.2	0.2	0.3	0.8	1.1	0.6	0.8
	RC25	146.1	0.2	0.6	130.5	0.2	0.6	191.3	0.2	0.6	0.8	1.1	0.7	1.0
	C50	205.4	0.3	1.9	171.6	0.2	1.7	252.6	0.3	1.7	0.8	1.1	0.7	1.0
	R50	251.9	0.4	2.3	214.7	0.3	2.3	364.6	0.5	2.3	0.7	1.0	0.6	1.0
	RC50	269.6	0.4	3.9	234.0	0.3	3.7	387.9	0.5	3.8	0.7	1.0	0.6	1.0
	C100	372.7	0.6	12.9	269.3	0.4	12.6	541.9	0.9	14.1	0.7	0.9	0.5	0.9
	R100	562.0	1.1	47.1	529.1	1.1	48.6	967.3	2.0	48.9	0.6	1.0	0.5	1.0
	RC100	443.9	0.8	47.7	407.3	0.8	52.0	740.5	1.4	49.8	0.6	1.0	0.5	1.0
	ALL	**281.7**	**0.5**	**13.2**	**245.7**	**0.4**	**13.7**	**428.1**	**0.7**	**13.8**	**0.7**	**1.0**	**0.6**	**1.0**
50	C25	114.1	0.2	0.3	114.9	0.2	0.4	152.7	0.3	0.4	0.7	0.8	0.8	1.0
	R25	108.0	0.2	0.3	105.8	0.1	0.3	138.0	0.2	0.3	0.8	0.8	0.8	0.9
	RC25	127.0	0.2	0.5	130.9	0.2	0.6	172.9	0.3	0.6	0.7	0.8	0.8	1.0
	C50	183.2	0.4	1.3	159.7	0.5	1.7	198.6	0.5	1.4	0.9	0.9	0.8	1.2
	R50	217.3	0.4	1.6	171.8	0.4	1.8	257.7	0.5	1.8	0.8	0.9	0.7	1.0
	RC50	225.0	0.4	2.5	204.3	0.5	2.9	269.9	0.5	2.7	0.8	0.9	0.8	1.1
	C100	354.8	0.9	10.1	314.2	1.1	11.0	468.2	1.5	10.6	0.8	0.9	0.7	1.0
	R100	464.3	1.5	28.8	411.8	2.1	36.9	708.8	2.7	30.6	0.7	0.9	0.6	1.2
	RC100	373.1	1.1	29.8	339.3	1.4	33.5	509.8	1.7	30.8	0.7	1.0	0.7	1.1
	ALL	**243.1**	**0.6**	**8.4**	**218.0**	**0.8**	**10.1**	**324.7**	**0.9**	**8.9**	**0.7**	**0.9**	**0.7**	**1.1**
100	C25	109.4	0.2	0.3	116.0	0.3	0.4	147.7	0.3	0.4	0.7	0.8	0.8	0.9
	R25	105.8	0.2	0.3	106.2	0.2	0.3	135.0	0.2	0.3	0.8	0.8	0.8	0.8
	RC25	130.1	0.2	0.5	141.0	0.3	0.6	162.4	0.3	0.6	0.8	0.9	0.9	1.0
	C50	169.9	0.5	1.6	144.0	0.6	1.7	207.1	0.7	1.7	0.8	1.0	0.7	1.0
	R50	187.8	0.4	1.4	161.9	0.5	1.6	255.2	0.7	1.6	0.7	0.9	0.6	0.9
	RC50	196.9	0.4	2.1	199.6	0.6	2.5	268.9	0.7	2.4	0.7	0.9	0.7	1.0
	C100	293.6	1.3	9.6	254.6	1.4	10.6	394.3	2.3	11.7	0.7	0.8	0.6	0.9
	R100	397.4	1.7	22.0	327.4	2.1	25.0	532.8	3.0	23.8	0.7	0.9	0.6	1.0
	RC100	337.4	1.3	22.4	280.0	1.6	17.4	419.3	2.0	22.7	0.8	1.0	0.7	0.8
	ALL	**215.7**	**0.7**	**6.8**	**192.4**	**0.8**	**6.9**	**283.0**	**1.2**	**7.3**	**0.8**	**0.9**	**0.7**	**0.9**
200	C25	128.0	0.3	0.4	116.8	0.4	0.5	151.9	0.4	0.5	0.8	0.8	0.8	0.9
	R25	104.3	0.2	0.3	108.7	0.2	0.3	124.4	0.3	0.4	0.8	0.8	0.9	0.9
	RC25	134.4	0.3	0.6	140.6	0.4	0.7	165.1	0.4	0.7	0.8	0.8	0.9	1.1
	C50	168.3	0.7	1.6	155.0	0.8	1.9	222.1	1.1	2.0	0.8	0.8	0.7	0.9
	R50	194.7	0.6	1.5	167.3	0.7	1.7	248.1	0.8	1.7	0.8	0.9	0.7	1.0
	RC50	195.0	0.6	2.3	221.9	1.0	2.8	282.1	1.0	2.6	0.7	0.9	0.8	1.1
	C100	323.0	2.2	10.1	261.6	2.3	9.9	348.8	2.8	10.0	0.9	1.0	0.7	1.0
	R100	394.3	2.5	19.8	315.9	3.0	21.8	510.6	4.2	21.1	0.8	0.9	0.6	1.0
	RC100	347.8	2.1	21.1	306.1	2.6	24.8	456.1	3.3	23.6	0.8	0.9	0.7	1.0
	ALL	**222.0**	**1.1**	**6.4**	**198.3**	**1.3**	**7.2**	**279.6**	**1.6**	**7.0**	**0.8**	**0.9**	**0.7**	**1.0**
300	C25	126.0	0.4	0.5	132.9	0.5	0.6	149.7	0.5	0.6	0.8	0.8	0.9	1.0
	R25	102.5	0.2	0.3	109.0	0.3	0.4	125.3	0.3	0.4	0.8	0.8	0.9	0.9
	RC25	138.6	0.4	0.7	151.6	0.5	0.8	176.5	0.6	0.8	0.8	0.9	0.9	0.9
	C50	186.2	1.1	2.1	157.9	1.2	2.1	208.1	1.2	2.0	0.9	1.1	0.8	1.0
	R50	194.8	0.7	1.6	173.0	0.9	1.8	270.6	1.1	2.0	0.7	0.8	0.6	0.9
	RC50	213.3	0.9	2.3	222.5	1.2	2.9	284.0	1.3	2.8	0.8	0.8	0.8	1.0
	C100	353.4	3.0	10.0	253.8	2.4	9.2	383.2	3.8	11.0	0.9	0.9	0.7	0.8
	R100	419.8	3.4	19.1	310.4	3.9	21.9	533.6	5.6	21.6	0.8	0.9	0.6	1.0
	RC100	347.0	2.5	20.1	284.9	3.1	22.4	468.9	3.9	21.5	0.7	0.9	0.6	1.0
	ALL	**232.0**	**1.4**	**6.3**	**198.6**	**1.6**	**7.0**	**290.3**	**2.1**	**7.0**	**0.8**	**0.9**	**0.7**	**1.0**

In Table 6.3 we present the results of our computational experience for VRPTW. As in Section 6.3.1, the first two columns denote the number of columns we aim to add per iteration to the RMP and the classification, respectively. We also present the average number of inner iterations (inner), the average RMP time in seconds (rmp(s)) and the average total time (tot(s)) per strategy and number of columns aimed to add per iteration (k).

As shown in Chapter 5, the most expensive task for this application in the PDCGM is solving the oracle (difference between the total and RMP times). When the number of columns aimed to be added per iteration is small, the RMP time accounts as a small proportion of the total CPU time. Therefore, it might seem a bit odd to see that for some classes the total CPU time is slightly lower while the RMP time is slightly higher. This is only due to savings in the number of outer iterations and therefore, in oracle time. The importance of the RMP time with respect to the total CPU time increases with the number of columns allowed to be added per iteration. This can be explained since the RMP becomes computationally more expensive (the instance becomes larger and the warmstarting strategy procedure, if any, consumes time) while the total CPU time is reduced.

Similarly, in Table 6.4 we include more information to help with the interpretation of our results, namely, the average number of outer iterations (ite), the average ratio between the time required to solve the RMPs and the number of outer iterations (rmp(s)/ite) and the average number of inner iterations per RMP (inner/ite).

Table 6.4: Average results on 87 instances of VRPTW for PFR, CS and 1SPDR adding k columns at a time: columns added, column generation calls and time per RMP.

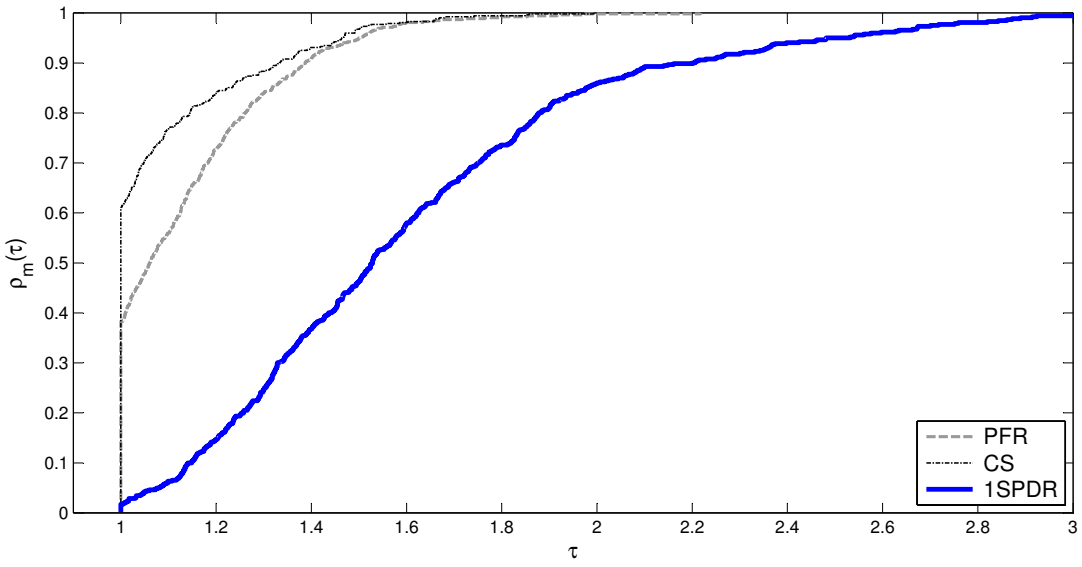
	PFR			**CS**			**1SPDR**		
k	**ite**	**rmp(s)/ite**	**inner/ite**	**ite**	**rmp(s)/ite**	**inner/ite**	**ite**	**rmp(s)/ite**	**inner/ite**
1	121.6	0.01	3.7	124.9	0.01	5.0	124.9	0.01	8.7
10	44.4	0.01	6.3	45.7	0.01	5.4	45.2	0.02	9.5
50	30.6	0.02	8.0	33.8	0.02	6.5	31.1	0.03	10.4
100	25.6	0.03	8.4	28.8	0.03	6.7	26.1	0.04	10.8
200	23.3	0.05	9.5	26.6	0.05	7.5	23.4	0.07	11.9
300	22.2	0.06	10.4	25.4	0.06	7.8	22.0	0.09	13.2

At first it seems surprising that CS performs so well. One explanation is that the presolving techniques used by HOPDM are very effective for this class of problems. Note that when one column is added, the average number of inner iterations per RMP for CS is 5.0 (column inner/ite in Table 6.4).

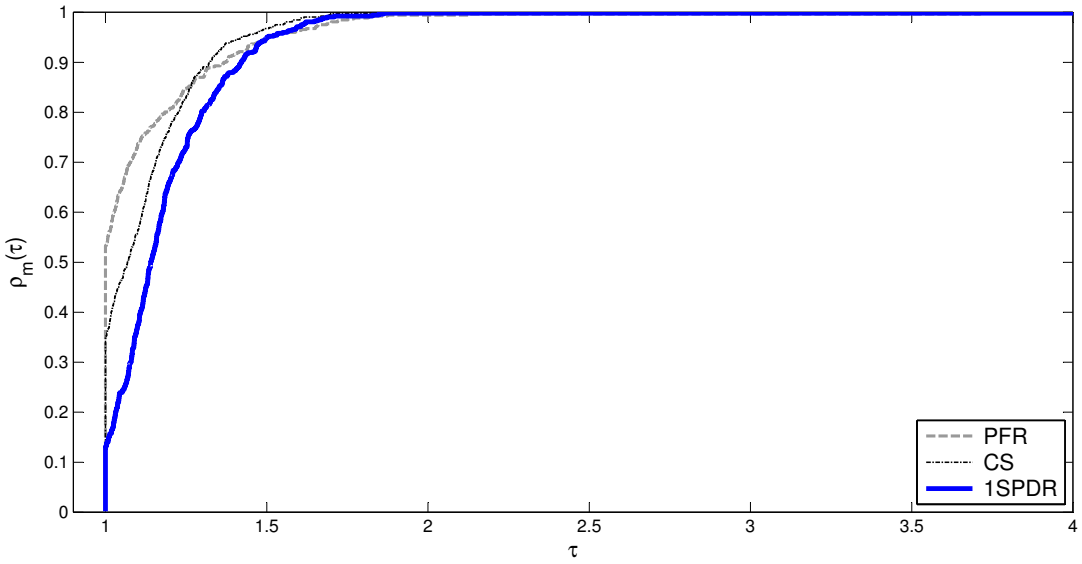
For this applications and the instances chosen, the depth of the cut is usually very large when compared with the reference parameter μ and therefore, the method retreats back in the list of iterates moving far away from the region close to optimality and therefore few more steps are required to reach the desirable sub-optimal solution of the modified problem.

However, the relative large number of inner iterations required for our strategy on average when compared to PFR and CS, does not have a significant impact on the efficiency of the method. If we compare the average time per iteration spent in solving the RMP, one can conclude that the RMP is solved very efficiently regardless the initial iterate (PFR, CS or 1SPDR). 1SPDR requires slightly more time to solve the RMP since this approach has to solve two auxiliary problems. The difficulty in solving the RMP increases in all cases with the number of columns added per iteration.

Similar to the results with CSP, the results for VRPTW show the robustness of the PDCGM in terms of outer iterations. Indeed, for the same number of columns added per iteration this figure remains almost constant, no matter if we use coldstart or any of the warmstarting strategies.



(a) Inner iterations



(b) Total CPU time (s)

Figure 6.4: Performance profiles for VRPTW with PFR, CS and 1SPDR

Additionally, in Figures 6.4(a) and 6.4(b) we have the performance profiles for VRPTW considering all the instances for all the values of k considered previously. The figures show the performance profiles for the inner iterations and total CPU time metrics obtained by the PDCGM when using PFR, CS and 1SPDR. Note that in terms of inner iterations, CS and PFR behave similarly while 1SPDR behaves poorly, being always outperformed by CS or PFR. Considering the total CPU time metric, the three strategies seem to provide similar results for the vast majority of instances. This supports the conclusion that for VRPTW all the reinitilization strategies used for the

PDCGM are more or less equally good.

It is important to point out that the columns obtained for VRPTW are very different from the ones obtained for CSP. While for CSP, the columns represent cutting patterns which have non-zero integer entries and may be dense, VRPTW has columns containing only few non-zero integer entries representing routes. Due to the vehicle capacity and time window constraints one may expect that only few customers are served by each vehicle so sparse columns are common. On the other hand for CSP, the more challenging instances are the ones with large width allowing more dense columns.

The structure of VRPTW adversely affects 1SPDR. It has been observed that if only few components of a new column have non-zero entries, the set of possible directions which recovers primal and dual feasibilities may be very limited. Therefore, large variations in small components are expected and as consequence of this, a bad performance of this strategy in practice.

Finally, from our computational experiments, one can conclude than in some situations, CS (understood as the initialization provided by HOPDM) may be competitive to an efficient warmstarting strategy. Nevertheless, we strongly believe that this is more an exception than the rule and therefore, warmstarting is indeed needed in this context. Note that for VRPTW all strategies studied here are efficient and solve each RMP in less than a tenth of a second.

Chapter 7

Conclusions and Further Developments

In this final chapter we summarise the main contributions of this thesis. We start with a summary of each chapter pointing out the key elements and giving some concluding remarks. Finally, we describe some avenues for future research.

7.1 Summary and concluding remarks

In this thesis we have presented a series of developments in order to combine efficiently column generation and interior point methods. Although both techniques have their own story of success in solving large linear programming problems, few attempts have been made to combine them.

In Chapter 2 we have described the primal-dual interior point method and the concept of neighbourhood of the central path, which is the driving force and the reason of the efficiency of this class of interior point methods. While the central path is the target that the primal-dual IPM follows to achieve convergence, the neighbourhood prevents the iterates from getting too close to the boundaries "too early". This is what keeps the iterates in the relative interior of the feasible set, allowing larger steps towards optimality, and only permitting iterates to get close to the boundaries when the duality gap is small. In this thesis we have considered the symmetric neighbourhood of the central path, $\mathcal{N}_s(\gamma)$, which bounds the complementarity products of the primal and dual slack variables from above and from below. We have shown that a feasible path-following method that uses the symmetric neighbourhood, keeps the same convergence and complexity results as if we were using the wide neighbourhood, $\mathcal{N}_{-\infty}$, converging in $O(n \log \frac{1}{\varepsilon})$ iterations, where ε is the tolerance and n the dimension of the problem.

In Chapter 3, we have described the Dantzig-Wolfe decomposition principle (DWD) and the column generation technique. We have also shown the equivalence of DWD and Lagrangian relaxation and justified the use of DWD over linear relaxation so that improved bounds are obtained. Additionally, we have discussed different cases one may encounter after using DWD such as aggregated and disaggregated master problem formulations and the addition of single or multiple columns per iteration. Also, we have described some well-known stabilization techniques to avoid the heading-in and tailing-off effects observed when the standard column generation method is applied.

In Chapter 4, we have presented three well-known problems in the context of combinatorial optimization, namely the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW) and the capacitated lot-sizing problem with

setup times (CLSPST), which can be tackled by DWD and column generation providing a tighter bound for a branch-and-bound (or branch-and-price) method. We have provided the standard formulation of these applications and derived the master problem and the subproblems formulations. These applications are standard problems in this context and provide different characteristics to test the new developments proposed in the following chapters.

In Chapter 5, we have presented the first developments in theory and new applications of the primal-dual column generation method (PDCGM). The technique relies on a primal-dual interior point method that obtains sub-optimal and well-centred solutions of the restricted master problems (RMP), leading to a more stable approach in relation to the standard column generation technique. The natural stabilization of the method is provided by the use of sub-optimal solutions delivered by the primal-dual interior point method together with the use of the symmetric neighbourhood. The PDCGM relies on an adaptive adjustment of the tolerance used to solve each restricted master problem (RMP). During the first stage of column generation, the RMPs are solved with a loose tolerance while in the second stage, when the column generation gap has closed enough, the method tightens this tolerance dynamically enforcing solutions closer to the optimal (Algorithm 5.1). A theoretical analysis is provided to show that the PDCGM converges to an optimum of the master problem if such exists, even though sub-optimal dual solutions are used in the course of the column generation process (Theorem 5.4). Additionally, computational experiments show that the method is competitive when compared with the standard column generation method (SCGM) and the analytic centre cutting plane method (ACCPM). These experiments were based on linear relaxations of integer master problems obtained from applying DWD to integer programming formulations of CSP, VRPTW and CLSPST. Different types of master problem formulations are obtained for these applications: an aggregated master problem in CSP with dense columns, an aggregated master problem with a set covering structure in VRPTW with sparse columns, and a disaggregated master problem in CLSPST. 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 SCGM and the ACCPM when solving large and difficult instances. Although the SCGM 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 with the SCGM gives an idea of how much can be gained by using sub-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 SCGM 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. Some additional comparisons are also discussed at the end of the chapter to show that some stabilization techniques neither speed up nor noticeably reduce the number of iterations when CSP and CLSPST are considered. Finally, for VRPTW, the computational evidence suggests that the efficiency of the PDCGM is comparable to that of the stabilization proposed in [109].

In Chapter 6 we have presented a new strategy to warmstart the PDCGM. Although the current strategy used by the PDCGM seems to be efficient, it lacks theoretical developments and no complete information about primal and dual feasibility nor centrality

of the proposed warmstarting iterate is available. In order to close this gap, we have designed a new warmstarting technique that aims to recover primal and dual feasibility while keeping control in the new complementarity products. This is done by solving two auxiliary linear programming problems which aim to minimise the change in small components in the primal and dual spaces. Once the direction calculated by these two auxiliary problems is available, a full step towards this direction is taken and we continue the primal-dual interior point iteration from this point (Algorithm 6.1). By avoiding large variations in small components we aim to perturb the complementarity products only slightly and, therefore, prevent any major increase in duality gap. We have provided evidence that the method recovers primal and dual feasibilities in one step after k new columns are appended to the RMP (Lemmas 6.4 and 6.6). Additionally, we have shown that the method obtains a new iterate that belongs to a slightly modified symmetric neighbourhood and therefore is well-centred (Theorem 6.7). Also, we have stated conditions to guarantee that the new duality gap is bounded by a constant and the old duality gap (Theorem 6.8). Despite the fact that the main objective of this part of the study was to develop a theoretical framework for warmstarting the PDCGM, we have also implemented the strategy to show its behaviour with respect to coldstart (CS) and the current warmstarting procedure used for the PDCGM (PFR) [56]. We have performed some computational experiments for solving a reformulation of CSP to demonstrate the benefits of using our warmstarting strategy (1SPDR) and compared it with CS. In general, savings between $30\% \sim 48\%$ in the number of inner iterations and $19\% \sim 45\%$ in CPU time can be achieved on average for different column generation scenarios. The advantages of the proposed method with respect to CS are consistent no matter the number of columns added to each restricted master problem at each iteration. Also, we have provided computational evidence that the proposed strategy and CS behave similarly in terms of CPU time for solving a reformulation of VRPTW, both strategies being very efficient due to the structure of the restricted master problems and the presolving embedded in the interior point solver, HOPDM. Additionally, we have compared our strategy with PFR which restores dual feasibility in the new components after new columns are added. Both warmstarting strategies are competitive in terms of CPU time for both applications, with 1SPDR slightly better for CSP and PFR slightly better for VRPTW. As observed from our computational experiments with CSP and VRPTW, it seems that the warmstarting strategies for the PDCGM are useful when dense columns are added to the RMP (CSP), since they consistently reduce CPU time and also the number of iterations required to solve the RMPs on average. On the other hand, when sparse columns are added (VRPTW) and the RMPs remain very simple, CS becomes very efficient and therefore warmstarting does not make the task of solving the RMPs any easier.

7.2 Further studies

Several avenues are available for further studies involving the primal-dual column generation technique and the warmstarting techniques presented here which were beyond the scope of this thesis but seem to be promising directions.

A possible direction of research is to combine the PDCGM with a branch-and-bound algorithm to obtain a branch-and-price or a branch-price-and-cut framework able to solve the original integer programming problems. This will require the development of several components such as branching rules, cutting plane methods to tighten the formulations at every node of the tree and a (slightly) different warmstarting technique.

A successful attempt to integrate some of these components has been proposed in [98], showing encouraging results for VRPTW where the idea of using well-centred and sub-optimal solutions in the PDCGM has been extended to the cut generation procedure, improving the general efficiency of the combined approach. Generalizations to other applications could be an avenue for further developments.

Also, a performance comparison of the PDCGM with advanced column generation methods such as bundle methods [43, 69, 79], the volume algorithm [6] and the BoxPen stabilization [32] for a similar selection of applications could be another extension of this research.

In a similar direction, we are currently investigating the performance of the PDCGM when solving large scale problems arising in several very different applications. In contrast with the computational study presented in Chapter 5, in which the applications' bottleneck were the subproblems which allowed us to solve medium size instances, we are interested in finding what the benefits of using the PDCGM in larger instances are, if any. So far, we have implemented the PDCGM to deal with *multicommodity network flow* (MCNF) [104], the *multiple kernel learning* (MKL) [113] and *two-stage stochastic programming* (TSS) [124] problems. Preliminary results show that the PDCGM outperforms several methods available in the literature for all these applications. We have compared the PDCGM with a wide variety of stabilized/unstabilized techniques, such as the analytic centre cutting plane method (ACCPM) [5], semi-infinite linear programming algorithm (unstabilized cutting plane method) [113], subgradient methods with a weighted l_2-norm regularization [105], Benders decomposition with regularisation by the level method [124], regularised decomposition [110] and the trust region method [83]. We believe that in addition to the efficiency of the method, the lack of extensive tuning and its straightforward description make the PDCGM an attractive option when solving problems by column generation/cutting plane methods. The new paper is in its final stage and will be submitted for publication shortly after the submission of this thesis.

In terms of warmstarting, there are some aspects that remain attractive for future research such as extensions of the current warmstarting strategy and some issues in its implementation.

Firstly, there is the issue of studying a warmstarting strategy that considers primal and dual infeasibilities so one could take advantage of the infeasible primal-dual interior point method solver HOPDM. It was noted in our computational experiments in Chapter 6 that the performance of PFR is favoured by the use of infeasible solutions which can be handled by the HOPDM. Therefore, providing theoretical guarantees of the centrality or convergence for such an infeasible method may be an attractive direction of research. Additionally, this might be a way to deal with sparse columns by allowing the method to have a larger pool of possible directions in order to generate a warmstarting mildly infeasible point.

Additionally, it was also observed that there is a trade-off for 1SPDR between the number of columns added per iteration and the time required to calculate the warmstarting iterate. For large values of k, solving two linear programming problems at every column generation iteration is an expensive task and therefore results in an excessively large CPU time. If we could bring down this time, it would have a great impact on the overall performance of the method. Hence, investigating more efficient ways of calculating these directions in the primal and dual spaces, bearing in mind the same objective, namely minimizing large perturbations of small components, seems to be a natural way of extending this study.

Moreover, studying how 1SPDR has to be modified so that it is able to calculate

a warmstarting iterate after some data perturbation, where the dimension of two consecutive problems remains unchanged, could be another possible direction of research.

Finally, it has been observed in Chapter 6 (and in [59]) that, although focusing on minimizing the change of the small components seems to be a reasonable way to proceed to avoid large variations in the complementarity products and centrality, one could achieve a similar objective by restricting the values of δ_l and δ_u which, together, bound the expansion of the symmetric neighbourhood of the central path, so these expansion factors are the minimum possible allowing the method to recover primal and dual feasibility. Undoubtedly, this new development requires the design of a new strategy for the PDCGM and, therefore, remains an attractive avenue for further developments.

Appendix A

Test problems

In this section we provide all the statistics for the instances used in this study. The description of each application can be found in Chapter 4. The number of constraints and variables were calculated using these descriptions without considering any presolving stage and are stated here as an indicative on how large the problems can be if they are tackled by a direct approach. Computational experiments using AMPL [42] and CPLEX [71] to solve the linear relaxation of the cutting stock problem and the vehicle routing problem with time windows, have shown that for large instances (over one million of variables and constraints), a direct approach struggles finding a solution. On the other hand, using column generation (particularly, the primal-dual variant), a solution is found in a reasonable time (see Chapter 5 and the computational experiments in Section 5.2).

A.1 Cutting stock problem

The benchmark instances for the one-dimensional cutting stock problem were obtained from `http://www.math.tu-dresden.de/~capad/`. In Tables A.1 and A.2, we show the statistics of each instance. For every instance we provide the name, class, number of rolls (m), number of customers (n), width (W), number of constraints (nbC) and number of integer variables (nbI).

Table A.1: CSP instance statistics - Set A

name	class	m	n	W	nbC	nbI
BPP_U09708_10292	U	585	1,463	30,000	2,048	857,318
BPP_U09723_10277	U	555	1,388	30,000	1,943	771,728
BPP_U09738_10262	U	525	1,313	30,000	1,838	690,638
BPP_U09753_10247	U	495	1,238	30,000	1,733	614,048
BPP_U09768_10232	U	465	1,163	30,000	1,628	541,958
BPP_U09783_10217	U	435	1,088	30,000	1,523	474,368
BPP_U09798_10202	U	405	1,013	30,000	1,418	411,278
BPP_U09813_10187	U	375	938	30,000	1,313	352,688
BPP_U09828_10172	U	345	863	30,000	1,208	298,598
BPP_U09843_10157	U	315	788	30,000	1,103	249,008
BPP_U09858_10142	U	285	713	30,000	998	203,918
BPP_U09873_10127	U	255	638	30,000	893	163,328
BPP_U09888_10112	U	225	563	30,000	788	127,238
BPP_U09903_10097	U	195	488	30,000	683	95,648
BPP_U09918_10082	U	165	413	30,000	578	68,558
BPP_U09933_10067	U	135	338	30,000	473	45,968
BPP_U09948_10052	U	105	263	30,000	368	27,878
BPP_U09963_10037	U	75	188	30,000	263	14,288
BPP_U09978_10022	U	45	113	30,000	158	5,198
BPP_U09993_10007	U	15	38	30,000	53	608

Continued on next page

Table A.1 – *Continued from previous page*

name	class	m	n	W	nbC	nbI
NN10	mX	200	76,071	100,000	76,271	15,290,271
NN1081	mX	199	93,607	100,000	93,806	18,721,400
NN1104	mX	199	141,246	100,000	141,445	28,249,200
NN1140	mX	200	56,223	100,000	56,423	11,300,823
NN121	mX	200	158,791	100,000	158,991	31,916,991
NN1223	mX	198	85,494	100,000	85,692	17,013,306
NN1234	mX	199	115,818	100,000	116,017	23,163,600
NN124	mX	199	69,454	100,000	69,653	13,890,800
NN1240	mX	200	80,189	100,000	80,389	16,117,989
NN1272	mX	200	95,160	100,000	95,360	19,127,160
NN1430	mX	200	66,201	100,000	66,401	13,306,401
NN1554	mX	199	60,289	100,000	60,488	12,057,800
NN1587	mX	198	60,847	100,000	61,045	12,108,553
NN1591	mX	200	187,973	100,000	188,173	37,782,573
NN1597	mX	200	94,873	100,000	95,073	19,069,473
NN170	mX	200	84,216	100,000	84,416	16,927,416
NN1788	mX	200	134,140	100,000	134,340	26,962,140
NN1853	mX	198	99,191	100,000	99,389	19,739,009
NN1898	mX	200	138,297	100,000	138,497	27,797,697
NN2110	mX	200	73,618	100,000	73,818	14,797,218
NN2408	mX	199	229,767	100,000	229,966	45,953,400
NN2662	mX	200	71,607	100,000	71,807	14,393,007
NN2710	mX	199	210,208	100,000	210,407	42,041,600
NN2727	mX	199	98,923	100,000	99,122	19,784,600
NN2784	mX	200	86,448	100,000	86,648	17,376,048
NN2815	mX	200	132,586	100,000	132,786	26,649,786
NN2911	mX	198	74,570	100,000	74,768	14,839,430
NN2916	mX	199	47,037	100,000	47,236	9,407,400
NN2935	mX	198	102,520	100,000	102,718	20,401,480
NN2971	mX	199	92,012	100,000	92,211	18,402,400
NN2978	mX	199	83,164	100,000	83,363	16,632,800
NN303	mX	200	50,049	100,000	50,249	10,059,849
NN3054	mX	200	75,744	100,000	75,944	15,224,544
NN3130	mX	199	73,649	100,000	73,848	14,729,800
NN3196	mX	199	44,324	100,000	44,523	8,864,800
NN3262	mX	199	164,998	100,000	165,197	32,999,600
NN3486	mX	199	96,402	100,000	96,601	19,280,400
NN3498	mX	199	124,920	100,000	125,119	24,984,000
NN352	mX	200	123,590	100,000	123,790	24,841,590
NN3534	mX	198	121,245	100,000	121,443	24,127,755
NN3535	mX	200	80,146	100,000	80,346	16,109,346
NN3536	mX	200	56,573	100,000	56,773	11,371,173
NN3570	mX	200	114,920	100,000	115,120	23,098,920
NN3582	mX	199	93,914	100,000	94,113	18,782,800
NN3623	mX	200	68,016	100,000	68,216	13,671,216
NN3675	mX	200	59,238	100,000	59,438	11,906,838
NN37	mX	199	67,043	100,000	67,242	13,408,600
NN3721	mX	200	101,520	100,000	101,720	20,405,520
NN3764	mX	200	115,881	100,000	116,081	23,292,081
NN3956	mX	200	139,638	100,000	139,838	28,067,238
NN4062	mX	198	196,674	100,000	196,872	39,138,126
NN4163	mX	200	181,036	100,000	181,236	36,388,236
NN4242	mX	199	75,696	100,000	75,895	15,139,200
NN4404	mX	199	85,478	100,000	85,677	17,095,600
NN4524	mX	200	93,276	100,000	93,476	18,748,476
NN4770	mX	200	149,194	100,000	149,394	29,987,994
NN4919	mX	199	91,612	100,000	91,811	18,322,400
NN4920	mX	200	95,706	100,000	95,906	19,236,906
NN5017	mX	200	245,725	100,000	245,925	49,390,725
NN5162	mX	200	84,798	100,000	84,998	17,044,398
NN5183	mX	200	76,530	100,000	76,730	15,382,530
NN5332	mX	200	68,043	100,000	68,243	13,676,643
NN535	mX	199	55,944	100,000	56,143	11,188,800
NN5384	mX	200	103,103	100,000	103,303	20,723,703
NN5415	mX	199	100,287	100,000	100,486	20,057,400
NN5468	mX	200	218,408	100,000	218,608	43,900,008
NN5587	mX	199	60,994	100,000	61,193	12,198,800
NN5662	mX	200	292,081	100,000	292,281	58,708,281

Continued on next page

Table A.1 – *Continued from previous page*

name	class	m	n	W	nbC	nbI
NN57	mX	199	77,463	100,000	77,662	15,492,600
NN5720	mX	199	72,290	100,000	72,489	14,458,000
NN5773	mX	200	58,987	100,000	59,187	11,856,387
NN5891	mX	200	50,306	100,000	50,506	10,111,506
NN6062	mX	200	54,494	100,000	54,694	10,953,294
NN609	mX	197	204,046	100,000	204,243	40,401,108
NN6134	mX	199	61,480	100,000	61,679	12,296,000
NN6192	mX	199	177,865	100,000	178,064	35,573,000
NN6260	mX	200	72,575	100,000	72,775	14,587,575
NN6293	mX	200	53,556	100,000	53,756	10,764,756
NN636	mX	199	100,344	100,000	100,543	20,068,800
NN6373	mX	199	56,426	100,000	56,625	11,285,200
NN6425	mX	197	324,902	100,000	325,099	64,330,596
NN6458	mX	199	72,968	100,000	73,167	14,593,600
NN6475	mX	200	59,658	100,000	59,858	11,991,258
NN6507	mX	199	138,457	100,000	138,656	27,691,400
NN6523	mX	200	111,888	100,000	112,088	22,489,488
NN6609	mX	199	219,955	100,000	220,154	43,991,000
NN6756	mX	200	69,918	100,000	70,118	14,053,518
NN678	mX	200	411,589	100,000	411,789	82,729,389
NN7110	mX	200	68,197	100,000	68,397	13,707,597
NN7251	mX	198	53,964	100,000	54,162	10,738,836
NN7407	mX	200	200,085	100,000	200,285	40,217,085
NN7413	mX	199	198,003	100,000	198,202	39,600,600
NN7501	mX	199	72,022	100,000	72,221	14,404,400
NN7507	mX	198	46,591	100,000	46,789	9,271,609
NN7522	mX	200	52,739	100,000	52,939	10,600,539
NN7524	mX	199	53,416	100,000	53,615	10,683,200
NN7553	mX	197	129,531	100,000	129,728	25,647,138
NN7587	mX	200	75,561	100,000	75,761	15,187,761
NN7588	mX	199	43,826	100,000	44,025	8,765,200
NN7715	mX	199	186,530	100,000	186,729	37,306,000
NN7731	mX	200	52,822	100,000	53,022	10,617,222
NN7760	mX	200	120,200	100,000	120,400	24,160,200
NN7776	mX	199	71,484	100,000	71,683	14,296,800
NN7790	mX	199	69,330	100,000	69,529	13,866,000
NN7815	mX	200	77,341	100,000	77,541	15,545,541
NN7919	mX	200	244,989	100,000	245,189	49,242,789
NN7942	mX	200	55,191	100,000	55,391	11,093,391
NN7965	mX	200	59,439	100,000	59,639	11,947,239
NN798	mX	200	107,848	100,000	108,048	21,677,448
NN7983	mX	200	58,892	100,000	59,092	11,837,292
NN8054	mX	200	63,994	100,000	64,194	12,862,794
NN8078	mX	199	106,792	100,000	106,991	21,358,400
NN8127	mX	199	93,394	100,000	93,593	18,678,800
NN8261	mX	199	162,417	100,000	162,616	32,483,400
NN8335	mX	200	86,666	100,000	86,866	17,419,866
NN8361	mX	200	170,327	100,000	170,527	34,235,727
NN837	mX	199	111,231	100,000	111,430	22,246,200
NN838	mX	200	241,088	100,000	241,288	48,458,688
NN8471	mX	200	106,700	100,000	106,900	21,446,700
NN8472	mX	199	242,035	100,000	242,234	48,407,000
NN850	mX	199	129,555	100,000	129,754	25,911,000
NN8555	mX	199	62,007	100,000	62,206	12,401,400
NN8559	mX	200	116,062	100,000	116,262	23,328,462
NN8560	mX	199	51,930	100,000	52,129	10,386,000
NN857	mX	198	104,521	100,000	104,719	20,799,679
NN8633	mX	200	66,920	100,000	67,120	13,450,920
NN8661	mX	199	288,945	100,000	289,144	57,789,000
NN8686	mX	199	110,487	100,000	110,686	22,097,400
NN87	mX	199	199,674	100,000	199,873	39,934,800
NN8724	mX	200	61,562	100,000	61,762	12,373,962
NN8727	mX	199	149,843	100,000	150,042	29,968,600
NN8741	mX	200	131,906	100,000	132,106	26,513,106
NN889	mX	200	68,025	100,000	68,225	13,673,025
NN8897	mX	199	121,053	100,000	121,252	24,210,600
NN8920	mX	199	90,251	100,000	90,450	18,050,200
NN90	mX	200	74,398	100,000	74,598	14,953,998

Continued on next page

Table A.1 – *Continued from previous page*

name	class	m	n	W	nbC	nbI
NN9082	mX	200	190,345	100,000	190,545	38,259,345
NN9313	mX	199	56,914	100,000	57,113	11,382,800
NN9380	mX	198	156,144	100,000	156,342	31,072,656
NN9589	mX	198	65,443	100,000	65,641	13,023,157
NN9637	mX	199	74,945	100,000	75,144	14,989,000
NN9660	mX	198	163,065	100,000	163,263	32,449,935
NN968	mX	200	155,119	100,000	155,319	31,178,919
NN9773	mX	199	130,609	100,000	130,808	26,121,800
NN9871	mX	199	116,138	100,000	116,337	23,227,600
18013075_003	MTP0xJES	110	348	1,000	458	38,628
18014074_072	MTP0xJES	114	336	1,000	450	38,640
20013072_054	MTP0xJES	115	366	1,000	481	42,456
16005085_033	MTP0	122	452	1,000	574	55,596
16005086_091	MTP0	124	478	1,000	602	59,750
18013075_003	MTP0	127	392	1,000	519	50,176
18014074_072	MTP0	124	360	1,000	484	45,000
20013072_054	MTP0	133	406	1,000	539	54,404
BPP119	hard28	173	1,376	1,000	1,549	239,424
BPP13	hard28	161	1,852	1,000	2,013	300,024
BPP14	hard28	136	1,008	1,000	1,144	138,096
BPP144	hard28	173	1,238	1,000	1,411	215,412
BPP175	hard28	185	1,160	1,000	1,345	215,760
BPP178	hard28	178	2,274	1,000	2,452	407,046
BPP181	hard28	157	1,034	1,000	1,191	163,372
BPP195	hard28	161	2,430	1,000	2,591	393,660
BPP359	hard28	164	895	1,000	1,059	147,675
BPP360	hard28	148	1,009	1,000	1,157	150,341
BPP40	hard28	144	965	1,000	1,109	139,925
BPP419	hard28	189	1,351	1,000	1,540	256,690
BPP47	hard28	158	1,276	1,000	1,434	202,884
BPP485	hard28	163	2,457	1,000	2,620	402,948
BPP531	hard28	175	1,265	1,000	1,440	222,640
BPP561	hard28	177	2,307	1,000	2,484	410,646
BPP60	hard28	144	546	1,000	690	79,170
BPP640	hard28	165	918	1,000	1,083	152,388
BPP645	hard28	141	1,766	1,000	1,907	250,772
BPP709	hard28	160	940	1,000	1,100	151,340
BPP716	hard28	158	1,353	1,000	1,511	215,127
BPP742	hard28	148	872	1,000	1,020	129,928
BPP766	hard28	143	807	1,000	950	116,208
BPP781	hard28	174	2,230	1,000	2,404	390,250
BPP785	hard28	163	1,153	1,000	1,316	189,092
BPP814	hard28	179	2,532	1,000	2,711	455,760
BPP832	hard28	139	972	1,000	1,111	136,080
BPP900	hard28	173	1,646	1,000	1,819	286,404
BPP_10000108_0257	7hard	85	494	1,000	579	42,484
BPP_12000107_0894	7hard	98	550	1,000	648	54,450
BPP_16000108_0149	7hard	123	759	1,000	882	94,116
BPP_18000108_0359	7hard	142	651	1,000	793	93,093
BPP_18000108_0716	7hard	129	721	1,000	850	93,730
BPP_20000108_0175	7hard	143	877	1,000	1,020	126,288
BPP_20005008_0124	7hard	141	564	1,000	705	80,088
BPP02015007_0918	53NIRUPs	20	45	1,000	65	945
BPP04000107_0697	53NIRUPs	38	194	1,000	232	7,566
BPP04005007_0612	53NIRUPs	40	130	1,000	170	5,330
BPP04005009_0361	53NIRUPs	39	115	1,000	154	4,600
BPP06000106_0782	53NIRUPs	58	287	1,000	345	16,933
BPP06000110_0195	53NIRUPs	58	583	1,000	641	34,397
BPP06005007_0033	53NIRUPs	56	234	1,000	290	13,338
BPP06005007_0382	53NIRUPs	57	238	1,000	295	13,804
BPP06005008_0755	53NIRUPs	58	220	1,000	278	12,980
BPP06015006_0067	53NIRUPs	58	143	1,000	201	8,437
BPP06015009_0403	53NIRUPs	57	134	1,000	191	7,772
BPP06015010_0771	53NIRUPs	58	114	1,000	172	6,726
BPP06025005_0639	53NIRUPs	52	139	1,000	191	7,367
BPP08000107_0624	53NIRUPs	73	439	1,000	512	32,486
BPP08005006_0862	53NIRUPs	75	323	1,000	398	24,548
BPP08005007_0394	53NIRUPs	72	304	1,000	376	22,192

Continued on next page

Table A.1 – *Continued from previous page*

name	class	m	n	W	nbC	nbI
BPP10000107_0508	53NIRUPs	92	678	1,000	770	63,054
BPP10000107_0733	53NIRUPs	93	1,290	1,000	1,383	121,260
BPP10000108_0257	53NIRUPs	92	837	1,000	929	77,841
BPP10015007_0464	53NIRUPs	91	224	1,000	315	20,608
BPP12000107_0894	53NIRUPs	109	1,567	1,000	1,676	172,370
BPP12000108_0035	53NIRUPs	113	916	1,000	1,029	104,424
BPP12000109_0917	53NIRUPs	115	555	1,000	670	64,380
BPP12000110_0655	53NIRUPs	119	666	1,000	785	79,920
BPP12005007_0138	53NIRUPs	112	332	1,000	444	37,516
BPP12005007_0180	53NIRUPs	112	468	1,000	580	52,884
BPP12005007_0324	53NIRUPs	109	431	1,000	540	47,410
BPP12005008_0450	53NIRUPs	112	337	1,000	449	38,081
BPP12015007_0165	53NIRUPs	109	278	1,000	387	30,580
BPP12015009_0740	53NIRUPs	112	240	1,000	352	27,120
BPP14005010_0447	53NIRUPs	134	333	1,000	467	44,955
BPP14015007_0121	53NIRUPs	129	291	1,000	420	37,830
BPP14015008_0373	53NIRUPs	123	310	1,000	433	38,440
BPP16000107_0014	53NIRUPs	136	1,008	1,000	1,144	138,096
BPP16000108_0149	53NIRUPs	146	1,140	1,000	1,286	167,580
BPP16000110_0279	53NIRUPs	144	1,266	1,000	1,410	183,570
BPP16005009_0001	53NIRUPs	141	434	1,000	575	61,628
BPP16005009_0161	53NIRUPs	150	419	1,000	569	63,269
BPP16015007_0206	53NIRUPs	142	346	1,000	488	49,478
BPP16015009_0981	53NIRUPs	141	348	1,000	489	49,416
BPP18000108_0359	53NIRUPs	164	895	1,000	1,059	147,675
BPP18000108_0716	53NIRUPs	158	1,353	1,000	1,511	215,127
BPP18000110_0439	53NIRUPs	162	1,310	1,000	1,472	213,530
BPP18015008_0821	53NIRUPs	161	374	1,000	535	60,588
BPP18015010_0605	53NIRUPs	162	350	1,000	512	57,050
BPP20000107_0119	53NIRUPs	173	1,376	1,000	1,549	239,424
BPP20000108_0025	53NIRUPs	175	1,800	1,000	1,975	316,800
BPP20000108_0175	53NIRUPs	185	1,160	1,000	1,345	215,760
BPP20005008_0124	53NIRUPs	173	634	1,000	807	110,316
BPP20005009_0200	53NIRUPs	175	608	1,000	783	107,008
BPP20005010_0953	53NIRUPs	184	640	1,000	824	118,400
BPP20015007_0637	53NIRUPs	161	466	1,000	627	75,492
BPP20015009_0272	53NIRUPs	178	365	1,000	543	65,335
GAU3	gau3	50	10,033	10,000	10,083	511,683

Table A.2: CSP instance statistics - Set B

name	m	n	W	nbC	nbI
BPP_U09498_10502	1,005	2,513	30,000	3,518	2,528,078
BPP_U09513_10487	975	2,438	30,000	3,413	2,379,488
BPP_U09528_10472	945	2,363	30,000	3,308	2,235,398
BPP_U09543_10457	915	2,288	30,000	3,203	2,095,808
BPP_U09558_10442	885	2,213	30,000	3,098	1,960,718
BPP_U09573_10427	855	2,138	30,000	2,993	1,830,128
BPP_U09588_10412	825	2,063	30,000	2,888	1,704,038
BPP_U09603_10397	795	1,988	30,000	2,783	1,582,448
BPP_U09618_10382	765	1,913	30,000	2,678	1,465,358
BPP_U09633_10367	735	1,838	30,000	2,573	1,352,768
BPP_U09648_10352	705	1,763	30,000	2,468	1,244,678
BPP_U09663_10337	675	1,688	30,000	2,363	1,141,088
BPP_U09678_10322	645	1,613	30,000	2,258	1,041,998
BPP_U09693_10307	615	1,538	30,000	2,153	947,408

A.2 Vehicle routing problem with time windows

For the vehicle routing problem with time windows we have considered 87 instances from the literature (`http://www2.imm.dtu.dk/~jla/solomon.html`), which were originally proposed in [112]. The name, class, number of customers (n), number of constraints (nbC), number of integer (nbI) and continuous (nbO) variables for every instance are

shown in Table A.3. For all the instances the capacity of the vehicle is 200.

Table A.3: VRPTW instance statistics - Set A

name	class	n	nbC	nbI	nbO	name	class	n	nbC	nbI	nbO
C101	C25	25	19,625	17,550	675	R107	R50	50	140,500	132,600	2,600
C102	C25	25	19,625	17,550	675	R108	R50	50	140,500	132,600	2,600
C103	C25	25	19,625	17,550	675	R109	R50	50	140,500	132,600	2,600
C104	C25	25	19,625	17,550	675	R110	R50	50	140,500	132,600	2,600
C105	C25	25	19,625	17,550	675	R111	R50	50	140,500	132,600	2,600
C106	C25	25	19,625	17,550	675	R112	R50	50	140,500	132,600	2,600
C107	C25	25	19,625	17,550	675	RC101	RC50	50	140,500	132,600	2,600
C108	C25	25	19,625	17,550	675	RC102	RC50	50	140,500	132,600	2,600
C109	C25	25	19,625	17,550	675	RC103	RC50	50	140,500	132,600	2,600
R101	R25	25	19,625	17,550	675	RC104	RC50	50	140,500	132,600	2,600
R102	R25	25	19,625	17,550	675	RC105	RC50	50	140,500	132,600	2,600
R103	R25	25	19,625	17,550	675	RC106	RC50	50	140,500	132,600	2,600
R104	R25	25	19,625	17,550	675	RC107	RC50	50	140,500	132,600	2,600
R105	R25	25	19,625	17,550	675	RC108	RC50	50	140,500	132,600	2,600
R106	R25	25	19,625	17,550	675	C101	C100	100	1,061,000	1,030,200	10,200
R107	R25	25	19,625	17,550	675	C102	C100	100	1,061,000	1,030,200	10,200
R108	R25	25	19,625	17,550	675	C103	C100	100	1,061,000	1,030,200	10,200
R109	R25	25	19,625	17,550	675	C104	C100	100	1,061,000	1,030,200	10,200
R110	R25	25	19,625	17,550	675	C105	C100	100	1,061,000	1,030,200	10,200
R111	R25	25	19,625	17,550	675	C106	C100	100	1,061,000	1,030,200	10,200
R112	R25	25	19,625	17,550	675	C107	C100	100	1,061,000	1,030,200	10,200
RC101	RC25	25	19,625	17,550	675	C108	C100	100	1,061,000	1,030,200	10,200
RC102	RC25	25	19,625	17,550	675	C109	C100	100	1,061,000	1,030,200	10,200
RC103	RC25	25	19,625	17,550	675	R101	R100	100	1,061,000	1,030,200	10,200
RC104	RC25	25	19,625	17,550	675	R102	R100	100	1,061,000	1,030,200	10,200
RC105	RC25	25	19,625	17,550	675	R103	R100	100	1,061,000	1,030,200	10,200
RC106	RC25	25	19,625	17,550	675	R104	R100	100	1,061,000	1,030,200	10,200
RC107	RC25	25	19,625	17,550	675	R105	R100	100	1,061,000	1,030,200	10,200
RC108	RC25	25	19,625	17,550	675	R106	R100	100	1,061,000	1,030,200	10,200
C101	C50	50	140,500	132,600	2,600	R107	R100	100	1,061,000	1,030,200	10,200
C102	C50	50	140,500	132,600	2,600	R108	R100	100	1,061,000	1,030,200	10,200
C103	C50	50	140,500	132,600	2,600	R109	R100	100	1,061,000	1,030,200	10,200
C104	C50	50	140,500	132,600	2,600	R110	R100	100	1,061,000	1,030,200	10,200
C105	C50	50	140,500	132,600	2,600	R111	R100	100	1,061,000	1,030,200	10,200
C106	C50	50	140,500	132,600	2,600	R112	R100	100	1,061,000	1,030,200	10,200
C107	C50	50	140,500	132,600	2,600	RC101	RC100	100	1,061,000	1,030,200	10,200
C108	C50	50	140,500	132,600	2,600	RC102	RC100	100	1,061,000	1,030,200	10,200
C109	C50	50	140,500	132,600	2,600	RC103	RC100	100	1,061,000	1,030,200	10,200
R101	R50	50	140,500	132,600	2,600	RC104	RC100	100	1,061,000	1,030,200	10,200
R102	R50	50	140,500	132,600	2,600	RC105	RC100	100	1,061,000	1,030,200	10,200
R103	R50	50	140,500	132,600	2,600	RC106	RC100	100	1,061,000	1,030,200	10,200
R104	R50	50	140,500	132,600	2,600	RC107	RC100	100	1,061,000	1,030,200	10,200
R105	R50	50	140,500	132,600	2,600	RC108	RC100	100	1,061,000	1,030,200	10,200
R106	R50	50	140,500	132,600	2,600						

Additionally, we have considered a set of more challenging instances with 200, 400 and 600 customers, which were proposed in [70]. The statistics of these benchmark instances are shown in Table A.4.

Table A.4: VRPTW instance statistics - Set B

name	n	nbC	nbI	nbO
R1_2_1	200	8,242,000	8,120,400	40,400
C1_2_1	200	8,242,000	8,120,400	40,400
RC1_2_1	200	8,242,000	8,120,400	40,400
R1_4_1	400	64,964,000	64,480,800	160,800
C1_4_1	400	64,964,000	64,480,800	160,800
RC1_4_1	400	64,964,000	64,480,800	160,800
R1_6_1	600	218,166,000	217,081,200	361,200
C1_6_1	600	218,166,000	217,081,200	361,200
RC1_6_1	600	218,166,000	217,081,200	361,200

A.3 Capacitated lot-sizing problem with setup times

For the capacitated lot-sizing problem with setup times we have modified 751 instances proposed in [114] by replicating each instance demand 5 times and dividing 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 A.5 and for every instance, we show the number of periods (n), the number of items (m), the number of constraints (nbC), the number of integer variables (nbI) and the number of continuous variables (nbO).

Table A.5: CLSPST instance statistics - Set A

name	class	n	m	nbC	nbI	nbO	name	class	n	m	nbC	nbI	nbO
E1	E	6	75	975	450	906	X12427A	X1	10	100	2,100	1,000	2,010
E10	E	6	75	975	450	906	X12427B	X1	10	100	2,100	1,000	2,010
E11	E	6	75	975	450	906	X12427C	X1	10	100	2,100	1,000	2,010
E12	E	6	75	975	450	906	X12427D	X1	10	100	2,100	1,000	2,010
E13	E	6	75	975	450	906	X12427E	X1	10	100	2,100	1,000	2,010
E14	E	6	75	975	450	906	X12428A	X1	10	100	2,100	1,000	2,010
E15	E	6	75	975	450	906	X12428B	X1	10	100	2,100	1,000	2,010
E16	E	6	75	975	450	906	X12428C	X1	10	100	2,100	1,000	2,010
E17	E	6	75	975	450	906	X12428D	X1	10	100	2,100	1,000	2,010
E18	E	6	75	975	450	906	X12428E	X1	10	100	2,100	1,000	2,010
E19	E	6	75	975	450	906	X12429A	X1	10	100	2,100	1,000	2,010
E2	E	6	75	975	450	906	X12429B	X1	10	100	2,100	1,000	2,010
E20	E	6	75	975	450	906	X12429C	X1	10	100	2,100	1,000	2,010
E21	E	6	75	975	450	906	X12429D	X1	10	100	2,100	1,000	2,010
E22	E	6	75	975	450	906	X12429E	X1	10	100	2,100	1,000	2,010
E23	E	6	75	975	450	906	X21117A	X2	20	100	4,100	2,000	4,020
E24	E	6	75	975	450	906	X21117B	X2	20	100	4,100	2,000	4,020
E25	E	6	75	975	450	906	X21117C	X2	20	100	4,100	2,000	4,020
E26	E	6	75	975	450	906	X21117D	X2	20	100	4,100	2,000	4,020
E27	E	6	75	975	450	906	X21117E	X2	20	100	4,100	2,000	4,020
E28	E	6	75	975	450	906	X21118A	X2	20	100	4,100	2,000	4,020
E29	E	6	75	975	450	906	X21118B	X2	20	100	4,100	2,000	4,020
E3	E	6	75	975	450	906	X21118C	X2	20	100	4,100	2,000	4,020
E30	E	6	75	975	450	906	X21118D	X2	20	100	4,100	2,000	4,020
E31	E	6	75	975	450	906	X21118E	X2	20	100	4,100	2,000	4,020
E32	E	6	75	975	450	906	X21119A	X2	20	100	4,100	2,000	4,020
E33	E	6	75	975	450	906	X21119B	X2	20	100	4,100	2,000	4,020
E34	E	6	75	975	450	906	X21119C	X2	20	100	4,100	2,000	4,020
E35	E	6	75	975	450	906	X21119D	X2	20	100	4,100	2,000	4,020
E36	E	6	75	975	450	906	X21119E	X2	20	100	4,100	2,000	4,020
E37	E	6	75	975	450	906	X21127A	X2	20	100	4,100	2,000	4,020
E38	E	6	75	975	450	906	X21127B	X2	20	100	4,100	2,000	4,020
E39	E	6	75	975	450	906	X21127C	X2	20	100	4,100	2,000	4,020
E4	E	6	75	975	450	906	X21127D	X2	20	100	4,100	2,000	4,020
E40	E	6	75	975	450	906	X21127E	X2	20	100	4,100	2,000	4,020
E41	E	6	75	975	450	906	X21128A	X2	20	100	4,100	2,000	4,020
E42	E	6	75	975	450	906	X21128B	X2	20	100	4,100	2,000	4,020
E43	E	6	75	975	450	906	X21128C	X2	20	100	4,100	2,000	4,020
E44	E	6	75	975	450	906	X21128D	X2	20	100	4,100	2,000	4,020
E45	E	6	75	975	450	906	X21128E	X2	20	100	4,100	2,000	4,020
E46	E	6	75	975	450	906	X21129A	X2	20	100	4,100	2,000	4,020
E47	E	6	75	975	450	906	X21129B	X2	20	100	4,100	2,000	4,020
E48	E	6	75	975	450	906	X21129C	X2	20	100	4,100	2,000	4,020
E49	E	6	75	975	450	906	X21129D	X2	20	100	4,100	2,000	4,020
E5	E	6	75	975	450	906	X21129E	X2	20	100	4,100	2,000	4,020
E50	E	6	75	975	450	906	X21217A	X2	20	100	4,100	2,000	4,020
E51	E	6	75	975	450	906	X21217B	X2	20	100	4,100	2,000	4,020
E53	E	6	75	975	450	906	X21217C	X2	20	100	4,100	2,000	4,020
E54	E	6	75	975	450	906	X21217D	X2	20	100	4,100	2,000	4,020
E55	E	6	75	975	450	906	X21217E	X2	20	100	4,100	2,000	4,020
E56	E	6	75	975	450	906	X21218A	X2	20	100	4,100	2,000	4,020
E57	E	6	75	975	450	906	X21218B	X2	20	100	4,100	2,000	4,020
E59	E	6	75	975	450	906	X21218C	X2	20	100	4,100	2,000	4,020

Continued on next page

Table A.5 – *Continued from previous page*

name	class	n	m	nbC	nbI	nbO	name	class	n	m	nbC	nbI	nbO
E6	E	6	75	975	450	906	X21218D	X2	20	100	4,100	2,000	4,020
E60	E	6	75	975	450	906	X21218E	X2	20	100	4,100	2,000	4,020
E7	E	6	75	975	450	906	X21219A	X2	20	100	4,100	2,000	4,020
E8	E	6	75	975	450	906	X21219B	X2	20	100	4,100	2,000	4,020
E9	E	6	75	975	450	906	X21219C	X2	20	100	4,100	2,000	4,020
F1	F	6	75	975	450	906	X21219D	X2	20	100	4,100	2,000	4,020
F10	F	6	75	975	450	906	X21219E	X2	20	100	4,100	2,000	4,020
F11	F	6	75	975	450	906	X21227A	X2	20	100	4,100	2,000	4,020
F12	F	6	75	975	450	906	X21227B	X2	20	100	4,100	2,000	4,020
F13	F	6	75	975	450	906	X21227C	X2	20	100	4,100	2,000	4,020
F14	F	6	75	975	450	906	X21227D	X2	20	100	4,100	2,000	4,020
F15	F	6	75	975	450	906	X21227E	X2	20	100	4,100	2,000	4,020
F16	F	6	75	975	450	906	X21228A	X2	20	100	4,100	2,000	4,020
F17	F	6	75	975	450	906	X21228B	X2	20	100	4,100	2,000	4,020
F18	F	6	75	975	450	906	X21228C	X2	20	100	4,100	2,000	4,020
F19	F	6	75	975	450	906	X21228D	X2	20	100	4,100	2,000	4,020
F2	F	6	75	975	450	906	X21228E	X2	20	100	4,100	2,000	4,020
F20	F	6	75	975	450	906	X21229A	X2	20	100	4,100	2,000	4,020
F21	F	6	75	975	450	906	X21229B	X2	20	100	4,100	2,000	4,020
F22	F	6	75	975	450	906	X21229C	X2	20	100	4,100	2,000	4,020
F23	F	6	75	975	450	906	X21229D	X2	20	100	4,100	2,000	4,020
F24	F	6	75	975	450	906	X21229E	X2	20	100	4,100	2,000	4,020
F25	F	6	75	975	450	906	X21417A	X2	20	100	4,100	2,000	4,020
F26	F	6	75	975	450	906	X21417B	X2	20	100	4,100	2,000	4,020
F27	F	6	75	975	450	906	X21417C	X2	20	100	4,100	2,000	4,020
F28	F	6	75	975	450	906	X21417D	X2	20	100	4,100	2,000	4,020
F29	F	6	75	975	450	906	X21417E	X2	20	100	4,100	2,000	4,020
F3	F	6	75	975	450	906	X21418A	X2	20	100	4,100	2,000	4,020
F30	F	6	75	975	450	906	X21418B	X2	20	100	4,100	2,000	4,020
F31	F	6	75	975	450	906	X21418C	X2	20	100	4,100	2,000	4,020
F32	F	6	75	975	450	906	X21418D	X2	20	100	4,100	2,000	4,020
F33	F	6	75	975	450	906	X21418E	X2	20	100	4,100	2,000	4,020
F34	F	6	75	975	450	906	X21419A	X2	20	100	4,100	2,000	4,020
F35	F	6	75	975	450	906	X21419B	X2	20	100	4,100	2,000	4,020
F36	F	6	75	975	450	906	X21419C	X2	20	100	4,100	2,000	4,020
F37	F	6	75	975	450	906	X21419D	X2	20	100	4,100	2,000	4,020
F38	F	6	75	975	450	906	X21419E	X2	20	100	4,100	2,000	4,020
F39	F	6	75	975	450	906	X21427A	X2	20	100	4,100	2,000	4,020
F4	F	6	75	975	450	906	X21427B	X2	20	100	4,100	2,000	4,020
F40	F	6	75	975	450	906	X21427C	X2	20	100	4,100	2,000	4,020
F41	F	6	75	975	450	906	X21427D	X2	20	100	4,100	2,000	4,020
F42	F	6	75	975	450	906	X21427E	X2	20	100	4,100	2,000	4,020
F43	F	6	75	975	450	906	X21428A	X2	20	100	4,100	2,000	4,020
F44	F	6	75	975	450	906	X21428B	X2	20	100	4,100	2,000	4,020
F45	F	6	75	975	450	906	X21428C	X2	20	100	4,100	2,000	4,020
F46	F	6	75	975	450	906	X21428D	X2	20	100	4,100	2,000	4,020
F47	F	6	75	975	450	906	X21428E	X2	20	100	4,100	2,000	4,020
F48	F	6	75	975	450	906	X21429A	X2	20	100	4,100	2,000	4,020
F49	F	6	75	975	450	906	X21429B	X2	20	100	4,100	2,000	4,020
F5	F	6	75	975	450	906	X21429C	X2	20	100	4,100	2,000	4,020
F50	F	6	75	975	450	906	X21429D	X2	20	100	4,100	2,000	4,020
F51	F	6	75	975	450	906	X21429E	X2	20	100	4,100	2,000	4,020
F52	F	6	75	975	450	906	X22117A	X2	20	100	4,100	2,000	4,020
F53	F	6	75	975	450	906	X22117B	X2	20	100	4,100	2,000	4,020
F54	F	6	75	975	450	906	X22117C	X2	20	100	4,100	2,000	4,020
F55	F	6	75	975	450	906	X22117D	X2	20	100	4,100	2,000	4,020
F56	F	6	75	975	450	906	X22117E	X2	20	100	4,100	2,000	4,020
F57	F	6	75	975	450	906	X22118A	X2	20	100	4,100	2,000	4,020
F58	F	6	75	975	450	906	X22118B	X2	20	100	4,100	2,000	4,020
F59	F	6	75	975	450	906	X22118C	X2	20	100	4,100	2,000	4,020
F6	F	6	75	975	450	906	X22118D	X2	20	100	4,100	2,000	4,020
F60	F	6	75	975	450	906	X22118E	X2	20	100	4,100	2,000	4,020
F61	F	6	75	975	450	906	X22119A	X2	20	100	4,100	2,000	4,020
F62	F	6	75	975	450	906	X22119B	X2	20	100	4,100	2,000	4,020
F63	F	6	75	975	450	906	X22119C	X2	20	100	4,100	2,000	4,020
F64	F	6	75	975	450	906	X22119D	X2	20	100	4,100	2,000	4,020
F65	F	6	75	975	450	906	X22119E	X2	20	100	4,100	2,000	4,020
F66	F	6	75	975	450	906	X22127A	X2	20	100	4,100	2,000	4,020

Continued on next page

Table A.5 – *Continued from previous page*

name	class	n	m	nbC	nbI	nbO	name	class	n	m	nbC	nbI	nbO
F67	F	6	75	975	450	906	X22127B	X2	20	100	4,100	2,000	4,020
F68	F	6	75	975	450	906	X22127C	X2	20	100	4,100	2,000	4,020
F69	F	6	75	975	450	906	X22127D	X2	20	100	4,100	2,000	4,020
F7	F	6	75	975	450	906	X22127E	X2	20	100	4,100	2,000	4,020
F70	F	6	75	975	450	906	X22128A	X2	20	100	4,100	2,000	4,020
F8	F	6	75	975	450	906	X22128B	X2	20	100	4,100	2,000	4,020
F9	F	6	75	975	450	906	X22128C	X2	20	100	4,100	2,000	4,020
W13	W	4	75	675	300	604	X22128D	X2	20	100	4,100	2,000	4,020
W14	W	4	75	675	300	604	X22128E	X2	20	100	4,100	2,000	4,020
W15	W	4	75	675	300	604	X22129A	X2	20	100	4,100	2,000	4,020
W16	W	4	75	675	300	604	X22129B	X2	20	100	4,100	2,000	4,020
W17	W	4	75	675	300	604	X22129C	X2	20	100	4,100	2,000	4,020
W18	W	4	75	675	300	604	X22129D	X2	20	100	4,100	2,000	4,020
W19	W	8	75	1,275	600	1,208	X22129E	X2	20	100	4,100	2,000	4,020
W20	W	8	75	1,275	600	1,208	X22217A	X2	20	100	4,100	2,000	4,020
W21	W	8	75	1,275	600	1,208	X22217B	X2	20	100	4,100	2,000	4,020
W22	W	8	75	1,275	600	1,208	X22217C	X2	20	100	4,100	2,000	4,020
W23	W	8	75	1,275	600	1,208	X22217D	X2	20	100	4,100	2,000	4,020
W24	W	8	75	1,275	600	1,208	X22217E	X2	20	100	4,100	2,000	4,020
G1	G	6	75	975	450	906	X22218A	X2	20	100	4,100	2,000	4,020
G10	G	6	75	975	450	906	X22218B	X2	20	100	4,100	2,000	4,020
G11	G	6	75	975	450	906	X22218C	X2	20	100	4,100	2,000	4,020
G13	G	6	75	975	450	906	X22218D	X2	20	100	4,100	2,000	4,020
G14	G	6	75	975	450	906	X22218E	X2	20	100	4,100	2,000	4,020
G16	G	6	75	975	450	906	X22219A	X2	20	100	4,100	2,000	4,020
G17	G	6	75	975	450	906	X22219B	X2	20	100	4,100	2,000	4,020
G18	G	6	75	975	450	906	X22219C	X2	20	100	4,100	2,000	4,020
G19	G	6	75	975	450	906	X22219D	X2	20	100	4,100	2,000	4,020
G2	G	6	75	975	450	906	X22219E	X2	20	100	4,100	2,000	4,020
G20	G	6	75	975	450	906	X22227A	X2	20	100	4,100	2,000	4,020
G21	G	6	75	975	450	906	X22227B	X2	20	100	4,100	2,000	4,020
G22	G	6	75	975	450	906	X22227C	X2	20	100	4,100	2,000	4,020
G23	G	6	75	975	450	906	X22227D	X2	20	100	4,100	2,000	4,020
G24	G	6	75	975	450	906	X22227E	X2	20	100	4,100	2,000	4,020
G25	G	6	75	975	450	906	X22228A	X2	20	100	4,100	2,000	4,020
G26	G	6	75	975	450	906	X22228B	X2	20	100	4,100	2,000	4,020
G27	G	6	75	975	450	906	X22228C	X2	20	100	4,100	2,000	4,020
G28	G	6	75	975	450	906	X22228D	X2	20	100	4,100	2,000	4,020
G29	G	6	75	975	450	906	X22228E	X2	20	100	4,100	2,000	4,020
G3	G	6	75	975	450	906	X22229A	X2	20	100	4,100	2,000	4,020
G30	G	6	75	975	450	906	X22229B	X2	20	100	4,100	2,000	4,020
G31	G	6	75	975	450	906	X22229C	X2	20	100	4,100	2,000	4,020
G32	G	6	75	975	450	906	X22229D	X2	20	100	4,100	2,000	4,020
G33	G	6	75	975	450	906	X22229E	X2	20	100	4,100	2,000	4,020
G34	G	6	75	975	450	906	X22417A	X2	20	100	4,100	2,000	4,020
G35	G	6	75	975	450	906	X22417B	X2	20	100	4,100	2,000	4,020
G36	G	6	75	975	450	906	X22417C	X2	20	100	4,100	2,000	4,020
G37	G	6	75	975	450	906	X22417D	X2	20	100	4,100	2,000	4,020
G38	G	6	75	975	450	906	X22417E	X2	20	100	4,100	2,000	4,020
G39	G	6	75	975	450	906	X22418A	X2	20	100	4,100	2,000	4,020
G4	G	6	75	975	450	906	X22418B	X2	20	100	4,100	2,000	4,020
G40	G	6	75	975	450	906	X22418C	X2	20	100	4,100	2,000	4,020
G42	G	6	75	975	450	906	X22418D	X2	20	100	4,100	2,000	4,020
G44	G	6	75	975	450	906	X22418E	X2	20	100	4,100	2,000	4,020
G45	G	6	75	975	450	906	X22419A	X2	20	100	4,100	2,000	4,020
G46	G	6	75	975	450	906	X22419B	X2	20	100	4,100	2,000	4,020
G47	G	6	75	975	450	906	X22419C	X2	20	100	4,100	2,000	4,020
G48	G	6	75	975	450	906	X22419D	X2	20	100	4,100	2,000	4,020
G49	G	6	75	975	450	906	X22419E	X2	20	100	4,100	2,000	4,020
G5	G	6	75	975	450	906	X22427A	X2	20	100	4,100	2,000	4,020
G50	G	6	75	975	450	906	X22427B	X2	20	100	4,100	2,000	4,020
G51	G	12	75	1,875	900	1,812	X22427C	X2	20	100	4,100	2,000	4,020
G52	G	12	75	1,875	900	1,812	X22427D	X2	20	100	4,100	2,000	4,020
G53	G	12	75	1,875	900	1,812	X22427E	X2	20	100	4,100	2,000	4,020
G54	G	12	75	1,875	900	1,812	X22428A	X2	20	100	4,100	2,000	4,020
G55	G	12	75	1,875	900	1,812	X22428B	X2	20	100	4,100	2,000	4,020
G56	G	24	75	3,675	1,800	3,624	X22428C	X2	20	100	4,100	2,000	4,020
G57	G	24	75	3,675	1,800	3,624	X22428D	X2	20	100	4,100	2,000	4,020

Continued on next page

Table A.5 – *Continued from previous page*

name	class	n	m	nbC	nbI	nbO	name	class	n	m	nbC	nbI	nbO
G58	G	24	75	3,675	1,800	3,624	X22428E	X2	20	100	4,100	2,000	4,020
G59	G	24	75	3,675	1,800	3,624	X22429A	X2	20	100	4,100	2,000	4,020
G6	G	6	75	975	450	906	X22429B	X2	20	100	4,100	2,000	4,020
G60	G	24	75	3,675	1,800	3,624	X22429C	X2	20	100	4,100	2,000	4,020
G61	G	6	150	1,950	900	1,806	X22429D	X2	20	100	4,100	2,000	4,020
G62	G	6	150	1,950	900	1,806	X22429E	X2	20	100	4,100	2,000	4,020
G63	G	6	150	1,950	900	1,806	X31117A	X3	30	100	6,100	3,000	6,030
G64	G	6	150	1,950	900	1,806	X31117B	X3	30	100	6,100	3,000	6,030
G65	G	6	150	1,950	900	1,806	X31117C	X3	30	100	6,100	3,000	6,030
G66	G	12	150	3,750	1,800	3,612	X31117D	X3	30	100	6,100	3,000	6,030
G67	G	12	150	3,750	1,800	3,612	X31117E	X3	30	100	6,100	3,000	6,030
G68	G	12	150	3,750	1,800	3,612	X31118A	X3	30	100	6,100	3,000	6,030
G69	G	12	150	3,750	1,800	3,612	X31118B	X3	30	100	6,100	3,000	6,030
G7	G	6	75	975	450	906	X31118C	X3	30	100	6,100	3,000	6,030
G70	G	12	150	3,750	1,800	3,612	X31118D	X3	30	100	6,100	3,000	6,030
G71	G	24	150	7,350	3,600	7,224	X31118E	X3	30	100	6,100	3,000	6,030
G72	G	24	150	7,350	3,600	7,224	X31119A	X3	30	100	6,100	3,000	6,030
G73	G	24	150	7,350	3,600	7,224	X31119B	X3	30	100	6,100	3,000	6,030
G74	G	24	150	7,350	3,600	7,224	X31119C	X3	30	100	6,100	3,000	6,030
G75	G	24	150	7,350	3,600	7,224	X31119D	X3	30	100	6,100	3,000	6,030
G8	G	6	75	975	450	906	X31119E	X3	30	100	6,100	3,000	6,030
G9	G	6	75	975	450	906	X31127A	X3	30	100	6,100	3,000	6,030
X11117A	X1	10	100	2,100	1,000	2,010	X31127B	X3	30	100	6,100	3,000	6,030
X11117B	X1	10	100	2,100	1,000	2,010	X31127C	X3	30	100	6,100	3,000	6,030
X11117C	X1	10	100	2,100	1,000	2,010	X31127D	X3	30	100	6,100	3,000	6,030
X11117D	X1	10	100	2,100	1,000	2,010	X31127E	X3	30	100	6,100	3,000	6,030
X11117E	X1	10	100	2,100	1,000	2,010	X31128A	X3	30	100	6,100	3,000	6,030
X11118A	X1	10	100	2,100	1,000	2,010	X31128B	X3	30	100	6,100	3,000	6,030
X11118B	X1	10	100	2,100	1,000	2,010	X31128C	X3	30	100	6,100	3,000	6,030
X11118C	X1	10	100	2,100	1,000	2,010	X31128D	X3	30	100	6,100	3,000	6,030
X11118D	X1	10	100	2,100	1,000	2,010	X31128E	X3	30	100	6,100	3,000	6,030
X11118E	X1	10	100	2,100	1,000	2,010	X31129A	X3	30	100	6,100	3,000	6,030
X11119A	X1	10	100	2,100	1,000	2,010	X31129B	X3	30	100	6,100	3,000	6,030
X11119B	X1	10	100	2,100	1,000	2,010	X31129C	X3	30	100	6,100	3,000	6,030
X11119C	X1	10	100	2,100	1,000	2,010	X31129D	X3	30	100	6,100	3,000	6,030
X11119D	X1	10	100	2,100	1,000	2,010	X31129E	X3	30	100	6,100	3,000	6,030
X11119E	X1	10	100	2,100	1,000	2,010	X31217A	X3	30	100	6,100	3,000	6,030
X11127A	X1	10	100	2,100	1,000	2,010	X31217B	X3	30	100	6,100	3,000	6,030
X11127B	X1	10	100	2,100	1,000	2,010	X31217C	X3	30	100	6,100	3,000	6,030
X11127C	X1	10	100	2,100	1,000	2,010	X31217D	X3	30	100	6,100	3,000	6,030
X11127D	X1	10	100	2,100	1,000	2,010	X31217E	X3	30	100	6,100	3,000	6,030
X11127E	X1	10	100	2,100	1,000	2,010	X31218A	X3	30	100	6,100	3,000	6,030
X11128A	X1	10	100	2,100	1,000	2,010	X31218B	X3	30	100	6,100	3,000	6,030
X11128B	X1	10	100	2,100	1,000	2,010	X31218C	X3	30	100	6,100	3,000	6,030
X11128C	X1	10	100	2,100	1,000	2,010	X31218D	X3	30	100	6,100	3,000	6,030
X11128D	X1	10	100	2,100	1,000	2,010	X31218E	X3	30	100	6,100	3,000	6,030
X11128E	X1	10	100	2,100	1,000	2,010	X31219A	X3	30	100	6,100	3,000	6,030
X11129A	X1	10	100	2,100	1,000	2,010	X31219B	X3	30	100	6,100	3,000	6,030
X11129B	X1	10	100	2,100	1,000	2,010	X31219C	X3	30	100	6,100	3,000	6,030
X11129C	X1	10	100	2,100	1,000	2,010	X31219D	X3	30	100	6,100	3,000	6,030
X11129D	X1	10	100	2,100	1,000	2,010	X31219E	X3	30	100	6,100	3,000	6,030
X11129E	X1	10	100	2,100	1,000	2,010	X31227A	X3	30	100	6,100	3,000	6,030
X11217A	X1	10	100	2,100	1,000	2,010	X31227B	X3	30	100	6,100	3,000	6,030
X11217B	X1	10	100	2,100	1,000	2,010	X31227C	X3	30	100	6,100	3,000	6,030
X11217C	X1	10	100	2,100	1,000	2,010	X31227D	X3	30	100	6,100	3,000	6,030
X11217D	X1	10	100	2,100	1,000	2,010	X31227E	X3	30	100	6,100	3,000	6,030
X11217E	X1	10	100	2,100	1,000	2,010	X31228A	X3	30	100	6,100	3,000	6,030
X11218A	X1	10	100	2,100	1,000	2,010	X31228B	X3	30	100	6,100	3,000	6,030
X11218B	X1	10	100	2,100	1,000	2,010	X31228C	X3	30	100	6,100	3,000	6,030
X11218C	X1	10	100	2,100	1,000	2,010	X31228D	X3	30	100	6,100	3,000	6,030
X11218D	X1	10	100	2,100	1,000	2,010	X31228E	X3	30	100	6,100	3,000	6,030
X11218E	X1	10	100	2,100	1,000	2,010	X31229A	X3	30	100	6,100	3,000	6,030
X11219A	X1	10	100	2,100	1,000	2,010	X31229B	X3	30	100	6,100	3,000	6,030
X11219B	X1	10	100	2,100	1,000	2,010	X31229C	X3	30	100	6,100	3,000	6,030
X11219C	X1	10	100	2,100	1,000	2,010	X31229D	X3	30	100	6,100	3,000	6,030
X11219D	X1	10	100	2,100	1,000	2,010	X31229E	X3	30	100	6,100	3,000	6,030
X11219E	X1	10	100	2,100	1,000	2,010	X31417A	X3	30	100	6,100	3,000	6,030
X11227A	X1	10	100	2,100	1,000	2,010	X31417B	X3	30	100	6,100	3,000	6,030

Continued on next page

Table A.5 – *Continued from previous page*

name	class	n	m	nbC	nbI	nbO	name	class	n	m	nbC	nbI	nbO
X11227B	X1	10	100	2,100	1,000	2,010	X31417C	X3	30	100	6,100	3,000	6,030
X11227C	X1	10	100	2,100	1,000	2,010	X31417D	X3	30	100	6,100	3,000	6,030
X11227D	X1	10	100	2,100	1,000	2,010	X31417E	X3	30	100	6,100	3,000	6,030
X11227E	X1	10	100	2,100	1,000	2,010	X31418A	X3	30	100	6,100	3,000	6,030
X11228A	X1	10	100	2,100	1,000	2,010	X31418B	X3	30	100	6,100	3,000	6,030
X11228B	X1	10	100	2,100	1,000	2,010	X31418C	X3	30	100	6,100	3,000	6,030
X11228C	X1	10	100	2,100	1,000	2,010	X31418D	X3	30	100	6,100	3,000	6,030
X11228D	X1	10	100	2,100	1,000	2,010	X31418E	X3	30	100	6,100	3,000	6,030
X11228E	X1	10	100	2,100	1,000	2,010	X31419A	X3	30	100	6,100	3,000	6,030
X11229A	X1	10	100	2,100	1,000	2,010	X31419B	X3	30	100	6,100	3,000	6,030
X11229B	X1	10	100	2,100	1,000	2,010	X31419C	X3	30	100	6,100	3,000	6,030
X11229C	X1	10	100	2,100	1,000	2,010	X31419D	X3	30	100	6,100	3,000	6,030
X11229D	X1	10	100	2,100	1,000	2,010	X31419E	X3	30	100	6,100	3,000	6,030
X11229E	X1	10	100	2,100	1,000	2,010	X31427A	X3	30	100	6,100	3,000	6,030
X11417A	X1	10	100	2,100	1,000	2,010	X31427B	X3	30	100	6,100	3,000	6,030
X11417B	X1	10	100	2,100	1,000	2,010	X31427C	X3	30	100	6,100	3,000	6,030
X11417C	X1	10	100	2,100	1,000	2,010	X31427D	X3	30	100	6,100	3,000	6,030
X11417D	X1	10	100	2,100	1,000	2,010	X31427E	X3	30	100	6,100	3,000	6,030
X11417E	X1	10	100	2,100	1,000	2,010	X31428A	X3	30	100	6,100	3,000	6,030
X11418A	X1	10	100	2,100	1,000	2,010	X31428B	X3	30	100	6,100	3,000	6,030
X11418B	X1	10	100	2,100	1,000	2,010	X31428C	X3	30	100	6,100	3,000	6,030
X11418C	X1	10	100	2,100	1,000	2,010	X31428D	X3	30	100	6,100	3,000	6,030
X11418D	X1	10	100	2,100	1,000	2,010	X31428E	X3	30	100	6,100	3,000	6,030
X11418E	X1	10	100	2,100	1,000	2,010	X31429A	X3	30	100	6,100	3,000	6,030
X11419A	X1	10	100	2,100	1,000	2,010	X31429B	X3	30	100	6,100	3,000	6,030
X11419B	X1	10	100	2,100	1,000	2,010	X31429C	X3	30	100	6,100	3,000	6,030
X11419C	X1	10	100	2,100	1,000	2,010	X31429D	X3	30	100	6,100	3,000	6,030
X11419D	X1	10	100	2,100	1,000	2,010	X31429E	X3	30	100	6,100	3,000	6,030
X11419E	X1	10	100	2,100	1,000	2,010	X32117A	X3	30	100	6,100	3,000	6,030
X11427A	X1	10	100	2,100	1,000	2,010	X32117B	X3	30	100	6,100	3,000	6,030
X11427B	X1	10	100	2,100	1,000	2,010	X32117C	X3	30	100	6,100	3,000	6,030
X11427C	X1	10	100	2,100	1,000	2,010	X32117D	X3	30	100	6,100	3,000	6,030
X11427D	X1	10	100	2,100	1,000	2,010	X32117E	X3	30	100	6,100	3,000	6,030
X11427E	X1	10	100	2,100	1,000	2,010	X32118A	X3	30	100	6,100	3,000	6,030
X11428A	X1	10	100	2,100	1,000	2,010	X32118B	X3	30	100	6,100	3,000	6,030
X11428B	X1	10	100	2,100	1,000	2,010	X32118C	X3	30	100	6,100	3,000	6,030
X11428C	X1	10	100	2,100	1,000	2,010	X32118D	X3	30	100	6,100	3,000	6,030
X11428D	X1	10	100	2,100	1,000	2,010	X32118E	X3	30	100	6,100	3,000	6,030
X11428E	X1	10	100	2,100	1,000	2,010	X32119A	X3	30	100	6,100	3,000	6,030
X11429A	X1	10	100	2,100	1,000	2,010	X32119B	X3	30	100	6,100	3,000	6,030
X11429B	X1	10	100	2,100	1,000	2,010	X32119C	X3	30	100	6,100	3,000	6,030
X11429C	X1	10	100	2,100	1,000	2,010	X32119D	X3	30	100	6,100	3,000	6,030
X11429D	X1	10	100	2,100	1,000	2,010	X32119E	X3	30	100	6,100	3,000	6,030
X11429E	X1	10	100	2,100	1,000	2,010	X32127A	X3	30	100	6,100	3,000	6,030
X12117A	X1	10	100	2,100	1,000	2,010	X32127B	X3	30	100	6,100	3,000	6,030
X12117B	X1	10	100	2,100	1,000	2,010	X32127C	X3	30	100	6,100	3,000	6,030
X12117C	X1	10	100	2,100	1,000	2,010	X32127D	X3	30	100	6,100	3,000	6,030
X12117D	X1	10	100	2,100	1,000	2,010	X32127E	X3	30	100	6,100	3,000	6,030
X12117E	X1	10	100	2,100	1,000	2,010	X32128A	X3	30	100	6,100	3,000	6,030
X12118A	X1	10	100	2,100	1,000	2,010	X32128B	X3	30	100	6,100	3,000	6,030
X12118B	X1	10	100	2,100	1,000	2,010	X32128C	X3	30	100	6,100	3,000	6,030
X12118C	X1	10	100	2,100	1,000	2,010	X32128D	X3	30	100	6,100	3,000	6,030
X12118D	X1	10	100	2,100	1,000	2,010	X32128E	X3	30	100	6,100	3,000	6,030
X12118E	X1	10	100	2,100	1,000	2,010	X32129A	X3	30	100	6,100	3,000	6,030
X12119A	X1	10	100	2,100	1,000	2,010	X32129B	X3	30	100	6,100	3,000	6,030
X12119B	X1	10	100	2,100	1,000	2,010	X32129C	X3	30	100	6,100	3,000	6,030
X12119C	X1	10	100	2,100	1,000	2,010	X32129D	X3	30	100	6,100	3,000	6,030
X12119D	X1	10	100	2,100	1,000	2,010	X32129E	X3	30	100	6,100	3,000	6,030
X12119E	X1	10	100	2,100	1,000	2,010	X32217A	X3	30	100	6,100	3,000	6,030
X12127A	X1	10	100	2,100	1,000	2,010	X32217B	X3	30	100	6,100	3,000	6,030
X12127B	X1	10	100	2,100	1,000	2,010	X32217C	X3	30	100	6,100	3,000	6,030
X12127C	X1	10	100	2,100	1,000	2,010	X32217D	X3	30	100	6,100	3,000	6,030
X12127D	X1	10	100	2,100	1,000	2,010	X32217E	X3	30	100	6,100	3,000	6,030
X12127E	X1	10	100	2,100	1,000	2,010	X32218A	X3	30	100	6,100	3,000	6,030
X12128A	X1	10	100	2,100	1,000	2,010	X32218B	X3	30	100	6,100	3,000	6,030
X12128B	X1	10	100	2,100	1,000	2,010	X32218C	X3	30	100	6,100	3,000	6,030
X12128C	X1	10	100	2,100	1,000	2,010	X32218D	X3	30	100	6,100	3,000	6,030
X12128D	X1	10	100	2,100	1,000	2,010	X32218E	X3	30	100	6,100	3,000	6,030

Continued on next page

Table A.5 – *Continued from previous page*

name	class	n	m	nbC	nbI	nbO	name	class	n	m	nbC	nbI	nbO
X12128E	X1	10	100	2,100	1,000	2,010	X32219A	X3	30	100	6,100	3,000	6,030
X12129A	X1	10	100	2,100	1,000	2,010	X32219B	X3	30	100	6,100	3,000	6,030
X12129B	X1	10	100	2,100	1,000	2,010	X32219C	X3	30	100	6,100	3,000	6,030
X12129C	X1	10	100	2,100	1,000	2,010	X32219D	X3	30	100	6,100	3,000	6,030
X12129D	X1	10	100	2,100	1,000	2,010	X32219E	X3	30	100	6,100	3,000	6,030
X12129E	X1	10	100	2,100	1,000	2,010	X32227A	X3	30	100	6,100	3,000	6,030
X12217A	X1	10	100	2,100	1,000	2,010	X32227B	X3	30	100	6,100	3,000	6,030
X12217B	X1	10	100	2,100	1,000	2,010	X32227C	X3	30	100	6,100	3,000	6,030
X12217C	X1	10	100	2,100	1,000	2,010	X32227D	X3	30	100	6,100	3,000	6,030
X12217D	X1	10	100	2,100	1,000	2,010	X32227E	X3	30	100	6,100	3,000	6,030
X12217E	X1	10	100	2,100	1,000	2,010	X32228A	X3	30	100	6,100	3,000	6,030
X12218A	X1	10	100	2,100	1,000	2,010	X32228B	X3	30	100	6,100	3,000	6,030
X12218B	X1	10	100	2,100	1,000	2,010	X32228C	X3	30	100	6,100	3,000	6,030
X12218C	X1	10	100	2,100	1,000	2,010	X32228D	X3	30	100	6,100	3,000	6,030
X12218D	X1	10	100	2,100	1,000	2,010	X32228E	X3	30	100	6,100	3,000	6,030
X12218E	X1	10	100	2,100	1,000	2,010	X32229A	X3	30	100	6,100	3,000	6,030
X12219A	X1	10	100	2,100	1,000	2,010	X32229B	X3	30	100	6,100	3,000	6,030
X12219B	X1	10	100	2,100	1,000	2,010	X32229C	X3	30	100	6,100	3,000	6,030
X12219C	X1	10	100	2,100	1,000	2,010	X32229D	X3	30	100	6,100	3,000	6,030
X12219D	X1	10	100	2,100	1,000	2,010	X32229E	X3	30	100	6,100	3,000	6,030
X12219E	X1	10	100	2,100	1,000	2,010	X32417A	X3	30	100	6,100	3,000	6,030
X12227A	X1	10	100	2,100	1,000	2,010	X32417B	X3	30	100	6,100	3,000	6,030
X12227B	X1	10	100	2,100	1,000	2,010	X32417C	X3	30	100	6,100	3,000	6,030
X12227C	X1	10	100	2,100	1,000	2,010	X32417D	X3	30	100	6,100	3,000	6,030
X12227D	X1	10	100	2,100	1,000	2,010	X32417E	X3	30	100	6,100	3,000	6,030
X12227E	X1	10	100	2,100	1,000	2,010	X32418A	X3	30	100	6,100	3,000	6,030
X12228A	X1	10	100	2,100	1,000	2,010	X32418B	X3	30	100	6,100	3,000	6,030
X12228B	X1	10	100	2,100	1,000	2,010	X32418C	X3	30	100	6,100	3,000	6,030
X12228C	X1	10	100	2,100	1,000	2,010	X32418D	X3	30	100	6,100	3,000	6,030
X12228D	X1	10	100	2,100	1,000	2,010	X32418E	X3	30	100	6,100	3,000	6,030
X12228E	X1	10	100	2,100	1,000	2,010	X32419A	X3	30	100	6,100	3,000	6,030
X12229A	X1	10	100	2,100	1,000	2,010	X32419B	X3	30	100	6,100	3,000	6,030
X12229B	X1	10	100	2,100	1,000	2,010	X32419C	X3	30	100	6,100	3,000	6,030
X12229C	X1	10	100	2,100	1,000	2,010	X32419D	X3	30	100	6,100	3,000	6,030
X12229D	X1	10	100	2,100	1,000	2,010	X32419E	X3	30	100	6,100	3,000	6,030
X12229E	X1	10	100	2,100	1,000	2,010	X32427A	X3	30	100	6,100	3,000	6,030
X12417A	X1	10	100	2,100	1,000	2,010	X32427B	X3	30	100	6,100	3,000	6,030
X12417B	X1	10	100	2,100	1,000	2,010	X32427C	X3	30	100	6,100	3,000	6,030
X12417C	X1	10	100	2,100	1,000	2,010	X32427D	X3	30	100	6,100	3,000	6,030
X12417D	X1	10	100	2,100	1,000	2,010	X32427E	X3	30	100	6,100	3,000	6,030
X12417E	X1	10	100	2,100	1,000	2,010	X32428A	X3	30	100	6,100	3,000	6,030
X12418A	X1	10	100	2,100	1,000	2,010	X32428B	X3	30	100	6,100	3,000	6,030
X12418B	X1	10	100	2,100	1,000	2,010	X32428C	X3	30	100	6,100	3,000	6,030
X12418C	X1	10	100	2,100	1,000	2,010	X32428D	X3	30	100	6,100	3,000	6,030
X12418D	X1	10	100	2,100	1,000	2,010	X32428E	X3	30	100	6,100	3,000	6,030
X12418E	X1	10	100	2,100	1,000	2,010	X32429A	X3	30	100	6,100	3,000	6,030
X12419A	X1	10	100	2,100	1,000	2,010	X32429B	X3	30	100	6,100	3,000	6,030
X12419B	X1	10	100	2,100	1,000	2,010	X32429C	X3	30	100	6,100	3,000	6,030
X12419C	X1	10	100	2,100	1,000	2,010	X32429D	X3	30	100	6,100	3,000	6,030
X12419D	X1	10	100	2,100	1,000	2,010	X32429E	X3	30	100	6,100	3,000	6,030
X12419E	X1	10	100	2,100	1,000	2,010							

Additionally, we have created instances in order to challenge the column generation methods. These instances were obtained by replicating each instance demand $5, 10, 15$ and 20 times and dividing the capacity, processing time, setup time and costs by the same factors. Also, we have increased the capacity by 10%. In Table A.6 the statistics of these instances are shown.

Table A.6: CLSPST instance statistics - Set B

name	n	m	nbC	nbI	nbO	name	n	m	nbC	nbI	nbO
t05G30	6	75	975	450	906	t15G30	6	225	2,925	1,350	2,706
t05G53	12	75	1,875	900	1,812	t15G53	12	225	5,625	2,700	5,412
t05G57	24	75	3,675	1,800	3,624	t15G57	24	225	11,025	5,400	10,824
t05X21117A	20	100	4,100	2,000	4,020	t15X21117A	20	300	12,300	6,000	12,020
t05X21117B	20	100	4,100	2,000	4,020	t15X21117B	20	300	12,300	6,000	12,020

Continued on next page

Table A.6 – *Continued from previous page*

name	n	m	nbC	nbI	nbO	name	n	m	nbC	nbI	nbO
t05X21118A	20	100	4,100	2,000	4,020	t15X21118A	20	300	12,300	6,000	12,020
t05X21118B	20	100	4,100	2,000	4,020	t15X21118B	20	300	12,300	6,000	12,020
t05X31117A	30	100	6,100	3,000	6,030	t15X31117A	30	300	18,300	9,000	18,030
t05X31117B	30	100	6,100	3,000	6,030	t15X31117B	30	300	18,300	9,000	18,030
t05X31118A	30	100	6,100	3,000	6,030	t15X31118A	30	300	18,300	9,000	18,030
t05X31118B	30	100	6,100	3,000	6,030	t15X31118B	30	300	18,300	9,000	18,030
t10G30	6	150	1,950	900	1,806	t20G30	6	300	3,900	1,800	3,606
t10G53	12	150	3,750	1,800	3,612	t20G53	12	300	7,500	3,600	7,212
t10G57	24	150	7,350	3,600	7,224	t20G57	24	300	14,700	7,200	14,424
t10X21117A	20	200	8,200	4,000	8,020	t20X21117A	20	400	16,400	8,000	16,020
t10X21117B	20	200	8,200	4,000	8,020	t20X21117B	20	400	16,400	8,000	16,020
t10X21118A	20	200	8,200	4,000	8,020	t20X21118A	20	400	16,400	8,000	16,020
t10X21118B	20	200	8,200	4,000	8,020	t20X21118B	20	400	16,400	8,000	16,020
t10X31117A	30	200	12,200	6,000	12,030	t20X31117A	30	400	24,400	12,000	24,030
t10X31117B	30	200	12,200	6,000	12,030	t20X31117B	30	400	24,400	12,000	24,030
t10X31118A	30	200	12,200	6,000	12,030	t20X31118A	30	400	24,400	12,000	24,030
t10X31118B	30	200	12,200	6,000	12,030	t20X31118B	30	400	24,400	12,000	24,030

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[2] A. Altman and K. C. Kiwiel. A note on some analytic center cutting plane methods for convex feasibility and minimization problems. *Computational Optimization and Applications*, 5(2):175–180, 1996.

[3] L. H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3(1):53–68, 1969.

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

[5] 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.

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

[7] C. Barnhart, A. Cohn, E. Johnson, D. Klabjan, G. Nemhauser, and P. Vance. Airline crew scheduling. In R. Hall, editor, *In Handbook in Transportation Science*, pages 517–560. Kluwer Academic Publishers, 2nd edition, 2003.

[8] 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.

[9] C. Beltran, C. Tadonki, and J. Vial. Solving the p-median problem with a semi-lagrangian relaxation. *Computational Optimization and Applications*, 35:239–260, 2006.

[10] 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.

[11] 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.

[12] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[13] H. Y. Benson and D. F. Shanno. An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming. *Computational Optimization and Applications*, 38(3):371–399, 2007.

[14] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

[15] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50:3–15, 2002.

[16] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[17] 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.

[18] F. Chung, M. Garey, and D. Johnson. On packing two-dimensional bins. *SIAM Journal of Algebraic and Discrete Methods*, pages 66–76, 1982.

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

[20] M. Colombo. *Advances in Interior Point Methods for Large-Scale Linear Programming*. PhD thesis, School of Mathematics, The University of Edinburgh, 2007.

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

[22] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):pp. 789–810, 1977.

[23] G. Dantzig. *Linear Programming And Extensions*. Princeton University Press, 1963.

[24] G. B. Dantzig and A. Madansky. On the solution of two-stage linear programs under uncertainty. In *Proceedings Fourth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 165 – 176. University of California Press, Berkeley, 1961.

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

[26] G. B. Dantzig and P. Wolfe. The decomposition algorithm for linear programs. *Econometrica*, 29(4):767–778, 1961.

[27] 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.

[28] 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.

[29] J. Desrosiers and M. E. Lübbecke. A primer in column generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 1–32. Springer US, 2005.

[30] I. Dikin. Iterative solution of problems of linear and quadratic programming. *Soviet Mathematics Doklady*, pages 674–675, 1967.

[31] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

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

[33] I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct methods for sparse matrices*. Oxford University Press, Inc., New York, NY, USA, 1986.

[34] S. Elhedhli and J.-L. Goffin. The integration of an interior-point cutting plane method within a branch-and-price algorithm. *Mathematical Programming*, 100(2):267–294, 2004.

[35] A. Engau, M. F. Anjos, and A. Vannelli. On interior-point warmstarts for linear and combinatorial optimization. *SIAM Journal on Optimization*, 20(4):1828–1861, 2010.

[36] D. Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, 1978.

[37] 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.

[38] FICO$^{\text{TM}}$Xpress Optimization Suite. *Xpress-Optimizer Reference Manual, Release 20.00*, 2009.

[39] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 50(12):1861–1871, 2004.

[40] J. Fliege. An efficient interior-point method for convex multicriteria optimization problems. *Mathematics of Operations Research*, 31(4):825–845, 2006.

[41] L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multicommodity network flows. *Management Science*, 5:97–101, 1958.

[42] R. Fourer, D. M. Gay, and B. W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 2002.

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

[44] R. M. Freund. A potential-function reduction algorithm for solving a linear program directly from an infeasible "warm start". *Mathematical Programming*, 52:441–466, 1991.

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

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

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

[48] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13(1):94–120, 1965.

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

[50] J.-L. Goffin, Z.-Q. Luo, and Y. Ye. Complexity analysis of an interior cutting plane method for convex feasibility problems. *SIAM Journal on Optimization*, 6(3):638–652, 1996.

[51] J.-L. Goffin and J.-P. Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. *Mathematical Programming*, 84:89–103, 1999.

[52] J.-L. Goffin and J.-P. Vial. Multiple cuts in the analytic center cutting plane method. *SIAM Journal on Optimization*, 11(1):266–288, 2000.

[53] 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.

[54] 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.

[55] J. Gondzio. Multiple centrality corrections in a primal-dual method for linear programming. *Computational Optimization and Applications*, 6(2):137–156, 1996.

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

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

[58] J. Gondzio, P. González-Brevis, and P. Munari. New developments in the primal-dual column generation technique. *European Journal of Operational Research*, 224(1):41–51, 2013.

[59] J. Gondzio and P. González-Brevis. A new warmstarting strategy for the primal-dual column generation method. Technical Report, ERGO 12-007, University of Edinburgh, School of Mathematics, 2012.

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

[61] 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.

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

[63] J. Gondzio and T. Terlaky. A computational view of interior-point methods for linear programming. In *Advances in linear and integer programming*, pages 103–144. University Press, 1994.

[64] C. C. Gonzaga. Path-following methods for linear programming. *SIAM Review*, 34(2):pp. 167–224, 1992.

[65] B. Gopalakrishnan and E. Johnson. Airline crew scheduling: state-of-the-art. *Annals of Operations Research*, 140:305–337, 2005.

[66] O. Güler, D. den Hertog, C. Roos, T. Terlaky, and T. Tsuchiya. Degeneracy in interior point methods for linear programming: A survey. *Annals of Operations Research*, 46(1):107–138, 1993.

[67] J. A. Hall. Towards a practical parallelisation of the simplex method. *Computational Management Science*, 7(2):139–170, 2010.

[68] J. A. Hall and K. I. McKinnon. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational Optimization and Applications*, 32(3):259–283, 2005.

[69] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. Springer-Verlag, 1993.

[70] 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.

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

[72] 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.

[73] 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.

[74] E. John and E. A. Yildirim. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications*, 41:151–183, 2008.

[75] 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.

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

[77] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

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

[79] K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical Programming*, 46:105–122, 1990.

[80] K. C. Kiwiel. Complexity of some cutting plane methods that use analytic centers. *Mathematical Programming*, pages 47–54, 1996.

[81] V. Klee and G. J. Minty. How good is the simplex algorithm? In O. Shisha, editor, *Inequalities - III*, pages 159–175. Academic Press, 1972.

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

[83] J. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications*, 24:207–250, 2003.

[84] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[85] A. Lodi, S. Martello, and D. Vigo. Recent advances on two-dimensional bin packing problems. *Discrete Applied Mathematics*, 123(1-3):379–396, 2002.

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

[87] I. Lustig, R. Marsten, and D. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM Journal on Optimization*, 2(3):435–449, 1992.

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

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

[90] MATLAB. *version 7.9.0.529 (R2009b)*. The MathWorks Inc., Natick, Massachusetts, 2009.

[91] N. Meggido. Pathways to the optimal set in linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming Interior-point and related methods*, pages 131–158. Springer-Verlag New York, Inc., USA, 1988.

[92] S. Mehrotra. On the implementation of a primal-dual interior point method. *SIAM Journal on Optimization*, 2(4):575–601, 1992.

[93] J. E. Mitchell. *Karmarkar's Algorithm and Combinatorial Optimization Problems*. PhD thesis, School of Operations Research and Industrial Engineering, Cornell University, 1988.

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

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

[96] J. E. 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.

[97] J. E. Mitchell and M. J. Todd. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, 56:245–284, 1992.

[98] P. Munari and J. Gondzio. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers & Operations Research*, 40(8):2026–2036, 2013.

[99] G. L. Nemhauser. Column generation for linear and integer programming. In M. Grötschel, editor, *Documenta Mathematica - Extra Volume ISMP*, pages 65–73. 2012.

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

[101] J. Nocedal and S. J.Wright. *Numerical Optimization*. Springer-Verlag, New York, 1999.

[102] M. A. Nunez and R. M. Freund. Condition measures and properties of the central trajectory of a linear program. *Mathematical Programming*, 83:1–28, 1998.

[103] M. R. Oskoorouchi, H. R. Ghaffari, T. Terlaky, and D. M. Aleman. An interior point constraint generation algorithm for semi-infinite optimization with healthcare application. *Operations Research*, 59(5):1184–1197, 2011.

[104] A. Ouorou, P. Mahey, and J.-P. Vial. A survey of algorithms for convex multicommodity flow problems. *Management Science*, 46(1):126–147, 2000.

[105] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 9:2491–2521, 2008.

[106] T. K. Ralphs and M. V. Galati. Decomposition and dynamic cut generation in integer linear programming. *Mathematical Programming*, 106:261–285, 2006.

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

[108] C. Roos, T. Terlaky, and J.-P. Vial. *Interior Point Methods for Linear Optimization*. Springer, 2nd edition, 2006.

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

[110] A. Ruszczyński. A regularized decomposition method for minimizing a sum of polyhedral functions. *Mathematical Programming*, 35:309–333, 1986.

[111] A. Skajaa, E. Andersen, and Y. Ye. Warmstarting the homogeneous and self-dual interior point method for linear and conic quadratic problems. *Mathematical Programming Computation*, 5:1–25, 2012.

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

[113] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006.

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

[115] F. Vanderbeck. *Decomposition and column generation for integer programs.* PhD thesis, Université Catholique de Louvain, Louvain la-Neuve, 1994.

[116] 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.

[117] 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.

[118] 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.

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

[120] P. Wentges. Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, 4(2):151–162, 1997.

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

[122] Y. Ye. *Interior-Point Algorithms: Theory and Analysis.* Series in Discrete Mathematics and Optimization. Wiley-Interscience, 1997.

[123] E. A. Yildirim and S. J. Wright. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, 12(3):782–810, 2002.

[124] V. Zverovich, C. Fábián, E. Ellison, and G. Mitra. A computational study of a solver system for processing two-stage stochastic LPs with enhanced Benders decomposition. *Mathematical Programming Computation*, 4:211–238, 2012.