Theoretical and computational issues for improving the performance of linear optimization methods

Pedro Augusto Munari Junior

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

Theoretical and computational issues for improving the performance of linear optimization methods[1]

Pedro Augusto Munari Junior

Advisor: **Prof. Marcos Nereu Arenales.**

Co-advisor: **Prof. Jacek Gondzio.**

Doctoral dissertation submitted to the *Instituto de Ciências Matemáticas e de Computação* - ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION.*

USP – São Carlos
April 2013

[1] This research was supported by FAPESP (São Paulo Research Foundation) and CAPES Foundation.

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

M963t

Munari Junior, Pedro Augusto
Theoretical and computational issues for improving the performance of linear optimization methods / Pedro Augusto Munari Junior; orientador Marcos Nereu Arenales; co-orientador Jacek Gondzio. -- São Carlos, 2013.
119 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2013.

1. Otimização linear. 2. Métodos tipo simplex. 3. Métodos de pontos interiores. 4. Geração de colunas. 5. Branch-price-and-cut. I. Arenales, Marcos Nereu, orient. II. Gondzio, Jacek, co-orient. III. Título.

Aspectos teóricos e computacionais para a melhoria do desempenho de métodos de otimização linear

Pedro Augusto Munari Junior

SERVIÇO DE PÓS-GRADUAÇÃO DO ICMC-USP

Data de Depósito:

Assinatura:_____

Aspectos teóricos e computacionais para a melhoria do desempenho de métodos de otimização linear[1]

Pedro Augusto Munari Junior

Orientador: **Prof. Marcos Nereu Arenales**

Co-orientador: **Prof. Jacek Gondzio**

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*.

USP – São Carlos
Abril de 2013

[1] Este trabalho foi financiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

M963t

Munari Junior, Pedro Augusto
Theoretical and computational issues for improving the performance of linear optimization methods / Pedro Augusto Munari Junior; orientador Marcos Nereu Arenales; co-orientador Jacek Gondzio. -- São Carlos, 2013.
119 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2013.

1. Otimização linear. 2. Métodos tipo simplex. 3. Métodos de pontos interiores. 4. Geração de colunas. 5. Branch-price-and-cut. I. Arenales, Marcos Nereu, orient. II. Gondzio, Jacek, co-orient. III. Título.

This thesis is dedicated to Alan Turing, the father of computer science.
His story shows how much the mankind lose with the prejudice.
As a scientist, his work goes beyond the machine.
As a man, his life goes beyond the human.

Acknowledgements

First of all, I thank God for giving me strength and perseverance to conduct my research. He was the only company in many sleepless nights and difficult times. Thanks to Him, I believed I could accomplish this doctoral degree. I am very thankful to my mother and my father for being amazing parents, giving me all the support and respecting my decisions. I am grateful to all my closest friends, who brings shine and happiness to my life. Special thanks to Douglas, Deise, Flavinha, and Gustavo. I am very grateful to Felipe, who has been a wonderful partner.

Many thanks to Prof. Arenales for guiding this research and for being so kind and supportive during all these years. I have learned from him not only about optimization, but also about life. I am very thankful to Prof. Jacek for accepting me as visitor at the University of Edinburgh and for being such a fantastic supervisor all the time. I really admire him as a lecturer, as a supervisor, as a friend and specially as a person. He taught me how to conduct my research seriously, and I am indebted to him because of that.

Thanks to the lecturers and professors of the ICMC/University of São Paulo. Special thanks to Prof. Franklina for helping me with several difficulties and for being so kind and respectful all the time. Thanks to all my friends and colleagues from the Optimization Laboratory (LOT) at the University of São Paulo, who made all the years we have spent together much happier and easier. A special thanks to Aline, Claudia, Douglas, Lana, Murilo, Pamela, Tamara and Victor, with whom I have shared most of my moments. I have learned a lot from them. Thanks to all my friends and colleagues from the University of Edinburgh as well, in special to Chris, George, Marina, Pablo and Pamela. They made my life in Scotland much warmer. I am also very thankful to Elizabeth, who was a very special friend in Edinburgh and is (fortunately) back in Brazil. I am also thankful to all the staff from the University of São Paulo and the University of Edinburgh.

I am grateful to the thesis examiners committee, namely Prof. Aurélio Ribeiro Oliveira, Prof. José Manuel Valério de Carvalho, Prof. Marcos Poggi de Aragão and Prof. Maristela Santos, for their valuable contributions to this thesis. I also thank the University of São Paulo and the Brazilian research funding councils *FAPESP (São Paulo Research Foundation)* and *CAPES Foundation* for all the support.

Abstract

Linear optimization tools are used to solve many problems that arise in our day-to-day lives. The linear optimization models and methodologies help to find, for example, the best amount of ingredients in our food, the most suitable routes and timetables for the buses and trains we take, and the right way to invest our savings. We would cite many other situations that involves linear optimization, since a large number of companies around the world base their decisions in solutions which are provided by the linear optimization methodologies. In this thesis, we propose theoretical and computational developments to improve the performance of important linear optimization methods. Namely, we address simplex type methods, interior point methods, the column generation technique and the branch-and-price method. In simplex-type methods, we investigate a variant which exploits special features of problems which are formulated in the general form. We present a novel theoretical description of the method and propose how to efficiently implement this method in practice. Furthermore, we propose how to use the primal-dual interior point method to improve the column generation technique. This results in the primal-dual column generation method, which is more stable in practice and has a better overall performance in relation to other column generation strategies. The primal-dual interior point method also offers advantageous features which can be exploited in the context of the branch-and-price method. We show that these features improves the branching operation and the generation of columns and valid inequalities. For all the strategies which are proposed in this thesis, we present the results of computational experiments which involves publicly available, well-known instances from the literature. The results indicate that these strategies help to improve the performance of the linear optimization methodologies. In particular for a class of problems, namely the vehicle routing problem with time windows, the interior point branch-and-price method proposed in this study was up to 33 times faster than a state-of-the-art implementation available in the literature.

Keywords: linear optimization; simplex type methods; interior point methods; column generation; branch-and-price.

Resumo

Ferramentas de otimização linear são usadas para resolver diversos problemas do nosso dia-a-dia. Os modelos e as metodologias de otimização linear ajudam a obter, por exemplo, a melhor quantidade de ingredientes na nossa alimentação, os horários e as rotas de ônibus e trens que tomamos, e a maneira certa para investir nossas economias. Muitas outras situações que envolvem otimização linear poderiam ser aqui citadas, já que um grande número de empresas em todo o mundo baseia suas decisões em soluções obtidas pelos métodos de otimização linear. Nesta tese, são propostos desenvolvimentos teóricos e computacionais para melhorar o desempenho de métodos de otimização linear. Em particular, serão abordados métodos tipo simplex, métodos de pontos interiores, a técnica de geração de colunas e o método *branch-and-price*. Em métodos tipo simplex, é investigada uma variante que explora as características especiais de problemas formulados na forma geral. Uma nova descrição teórica do método é apresentada e, também, são propostas técnicas computacionais para a implementação eficiente do método. Além disso, propõe-se como utilizar o método primal-dual de pontos interiores para melhorar a técnica de geração de colunas. Isto resulta no método primal-dual de geração de colunas, que é mais estável na prática e tem melhor desempenho geral em relação a outras estratégias de geração de colunas. O método primal-dual de pontos interiores também oferece características vantajosas que podem ser exploradas em conjunto com o método branch-and-price. De acordo com a investigação realizada, estas características melhoram a operação de ramificação e a geração de colunas e de desigualdades válidas. Para todas as estratégias propostas neste trabalho, são apresentados os resultados de experimentos computacionais envolvendo problemas de teste bem conhecidos e disponíveis publicamente. Os resultados indicam que as estratégias propostas ajudam a melhorar o desempenho das metodologias de otimização linear. Em particular para uma classe de problemas, o problema de roteamento de veículos com janelas de tempo, o método *branch-and-price* de pontos interiores proposto neste estudo foi até 33 vezes mais rápido que uma implementação estado-da-arte disponível na literatura.

Palavras-chave: Otimização linear; métodos tipo simplex; métodos de pontos interiores; geração de colunas, *branch-and-price*.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Information systems have evolved from simple data-management softwares to complex frameworks. Currently, some of them are able to execute very difficult tasks such as finding the best timetable for a public transportation system, suggesting a disease treatment, and forecasting natural disasters. To achieve such status, the advances in hardware and software engineering are important, as they allow computers to be more and more efficient after each year. But, in addition to that, complex systems rely on strong theoretical developments in mathematics, physics and related sciences. These developments allow real-life problems to be represented by mathematical models which are suitable for cleverly designed algorithms. Linear optimization is a successful example of a theoretical subject that is nowadays crucial to many complex information systems.

By linear optimization we refer to the theory and methods related to minimizing (or maximizing) a given linear function, which is restricted to a domain that is fully described by a set of linear equations or inequalities. Many real-life problems can be represented this way, in particular those faced by manufacturing companies, transportation and telecommunication businesses, and financial markets. The key point in the linear optimization is that the problems can be formulated by a relatively simple mathematical language, and powerful computational methods are available to tackle these formulations.

The linear optimization is typically divided into two main subjects: *linear programming* and *integer programming*. In the former, the domain of the function we are interested in optimizing is continuous. It is useful when we are looking for solutions which can be stated as fractional amounts, *e.g.*, when we want to know how much money to invest in a project, or what is the best proportion of ingredients in a blending. On the other hand, the integer programming is used when problems require decisions that can only be represented by integer values, so the domain of the function must be discrete. This case involves the search for decisions such as how many vehicles a company should use to deliver goods, which region is the best for setting a warehouse. It can also involve cases in which the solutions have discrete as well as fractional decisions, *e.g.* in a set of projects to invest, which one we should select and how much money we should assign to them.

The linear programming ideas started to be formalized during the Second World War, due to the need for efficient allocation of resources. The subject called the attention of the community when George B. Dantzig proposed the simplex method in 1947, a practical algorithm to solve linear programming formulations (Dantzig, 1951). With this method, small-size problems were solved even by using calculators. The first computational implementations of the method were very promising and motivated the investigation of efficient techniques to improve the performance of the method (Dantzig and Orchard-Hays, 1954; Orchard-Hays, 1968; Maros, 2003a). Until the beginning of the Eighties, simplex-type methods were still the

only practical methods available to solving linear programming problems. Then, the projective interior point method proposed by Karmarkar (1984) started a new age in linear programming. It was the first polynomial time algorithm for linear programming that was announced to be efficient in practice as well. Previous to Karmarkar's method, Khachiyan (1979) had proposed an algorithm for linear programming with polynomial complexity as well, but with poor computational performance in practice. The theoretical and practical issues of the interior point methods used nowadays are different from those presented by Karmarkar. Nevertheless, the author is recognized for having called the attention of the optimization community to a new methodology, which stimulated an intense research in the area. The developments in interior point methods stimulated at the same time the investigation for efficient techniques to improve simplex-type methods, in order to keep these methods competitive in practice. As a consequence, important achievements were accomplished for these linear programming methodologies, and the methods became able to solve larger and larger problems (Bixby et al., 1992; Bixby, 2002; Maros, 2003a; Gondzio and Grothey, 2006; Gondzio, 2012).

Currently, simplex-type methods and interior point methods are the dominant linear programming methodologies. These are powerful general-purpose methods that are available in the majority of the optimization softwares as well as information systems that need to optimize linear models. The free software COIN-OR (COIN-OR Foundation, Inc., 2012) and the commercial software CPLEX (IBM ILOG CPLEX v.12.1, 2010) are examples of packages which offer efficient implementations of the simplex method, in its primal and dual versions, and the primal-dual interior point method. It is important to have both methods available because they may show different performances on different classes of problems. The best method depends on issues related to the dimensions of the model, the structure and sparsity of the coefficient matrix, among others. As a consequence, it is difficult to ensure in advance which method will be the best for an specific problem. See Bixby (2002) for a computational study that compares implementations of these methods.

The integer programming field emerged from the need to obtain solutions with integer valued amounts. The simplex method was able to optimize general-purpose linear formulations that were restricted to continuous domains only. Until the mid-Fifties, there was no computational method able to guarantee optimal integer solutions to general-purpose integer programming models. Fortunately, this scenario changed with the introduction of the cutting plane method by Gomory (1958). The strategy was based on iteratively inserting *cuts* (linear inequalities) to the formulation after solving it by the simplex method. A couple of years later, Land and Doig (1960) came up with a completely different approach for obtaining optimal solutions for integer programming problems: the *branch-and-bound* method. Interestingly enough, these two methods are still nowadays the most used to solve integer programming formulations. The combination of these two approaches results in the *branch-and-cut* method, which is currently the only general-purpose integer programming methodology available in optimization softwares. For instance, powerful implementations of the branch-and-cut method are available in the softwares cited above (COIN-OR and CPLEX).

Another important integer programming methodology is the *branch-and-price* method. It consists in the combination of the branch-and-bound method with a linear programming methodology that is called the *column generation* method. Ford and Fulkerson (1958) proposed the column generation method motivated by a linear programming problem which was too large to be stored in the computer memory. Since then, the column generation has been applied to solve many problems, specially when combined with the branch-and-bound method. Differently form the branch-and-cut method, the implementation of a branch-and-price method is specific to solve a given problem. General-purpose implementations of this method are currently experimental (Galati et al., 2012; Lübbecke, 2012).

In this thesis, we investigate new strategies to improve the computational performance of linear optimization methodologies. In linear programming, we present contributions regarding its two main methodologies. We address a variant of the dual simplex method which shows better performance on formulations which have lower and upper bounds for variables and constraints. In addition, we show the advantageous features of using the primal-dual interior point method within the column generation method. This research also contributes with integer programming methodologies. In particular, we propose how to improve the performance of a branch-price-and-cut method by using the primal-dual interior point algorithm. To verify the behavior of the strategies that are proposed in this thesis, we use benchmarking instances that are available in public repositories and have been widely used in other researches. Besides, classical integer programming problems are addressed in the computational experiments. In a class of problems, namely the vehicle routing problem with time windows, the interior point branch-price-and-cut method proposed in this study was up to 33 times faster than a state-of-the-art implementation of a standard branch-price-and-cut.

The remaining chapters in this thesis have the following description:

- Chapter 2: we briefly describe the main linear programming methodologies, namely simplex-type methods and the interior point methods. The purpose is to introduce the main aspects of each method and setting up the notation and background information for the remaining chapters. In addition, we propose a unified framework to describe simplex type methods and interior point methods. This framework helps to show the differences as well as the similarities regarding these methods. We also briefly discuss the main ideas which are used to warmstart the primal-dual interior point method.

- Chapter 3: we further investigate a variant of the dual simplex method, which is designed for problems formulated in the general form. We present a novel theoretical description of the method, which extends the results of previous studies. Furthermore, we describe computational techniques that lead to the efficient implementation of the method. To verify the performance of the proposed variants, we present the main results of computational experiments using the Netlib instances.

- Chapter 4: we address the use of the primal-dual interior point algorithm within the column generation method. The motivation is given by the well-centered dual solutions that are offered by the interior point method. These solutions are more stable than the optimal dual solutions that are often obtained by a simplex-type algorithm in the standard column generation method. The resulting variant, which we call as the primal-dual column generation method, shows a better overall performance when compared to the standard column generation. We prove the finite convergence of the method and show the results of extensive computational experiments. The performance of the proposed approach is compared against the performance of two other variants of the column generation method. We used the linear relaxations of three classical integer programming problems in the experiments, namely the cutting stock problem, the vehicle routing problem with time windows, and the capacitated lot sizing problem with setup times. The results indicate that the proposed variant achieves the best overall performance regarding number of iterations and total CPU time in relation to other column generation variants from the literature. In addition, the relative performance of the method was even better for classes of larger instances.

- Chapter 5: we propose how to improve the performance of a branch-price-and-cut method by using advantageous features that are offered by the primal-dual interior

point algorithm. In particular, two features are important in this context: (*i*) the well-centered primal and dual solutions offered by the interior point method are beneficial to generate columns and valid inequalities; (*ii*) the early termination of the method leads to suboptimal solutions that are good approximations of optimal solutions, which contributes with the branching operation and with the generation of valid inequalities as well. We discuss in detail how to deal with the challenges of integrating the interior point algorithm with each core component of the branch-price-and-cut method. The performance of the proposed interior point branch-price-and-cut method was verified in well-known instances of the vehicle routing problem with time windows. The results confirm that the proposed approach is robust and delivers the best overall performance when compared against the results of a state-of-the-art standard branch-price-and-cut method available in the literature.

The aim of this research is to contribute with the state-of-the-art in linear optimization methodologies. Furthermore, it allows the improvement of the information systems that are based on linear optimization methodologies and, hence, it can be beneficial to organizations that rely on these systems to make decisions. The simplex method was recognized as one of the ten most influential algorithms to sciences and engineering (Dongarra and Sullivan, 2000). The literature estimates that the simplex method is called around thousands of times per second in softwares around the world (Elwes, 2012). Interior point methods are not far from this usage, and they have the additional advantage of being also applied to problems that cannot be modeled by means of linear functions only (Nemirovski and Todd, 2008; Gondzio, 2012). In other words, interior point methods can be used to solve a broader range of models, in which more complex features can be absorbed in the model by using, *e.g.*, quadratic and logarithmic functions. Finally, the column generation and the branch-and-price methods are currently the only methodologies which are able to find optimal solutions, in a reasonable amount of time, of important classes of problems in areas such as manufacturing and logistics (Lübbecke and Desrosiers, 2005; Desrosiers and Lübbecke, 2010).

Chapter 2

Linear programming methodologies

Linear programming is important for solving problems that emerges from different situations. It offers intuitive formulations which are based on linear functions and linear systems of equations/inequalities. Currently, these formulations can be solved by efficient implementations of cleverly designed linear programming methods. In addition, the linear programming methodologies are often used as auxiliary tools for solving more complex problems. For instance, most of the integer programming methodologies rely on linear programming to solve relaxations of the problem. The solutions which are obtained from the relaxations guide the search for an optimal integer solution. Nonlinear programming and nondifferentiable optimization are also examples of areas that demand the linear programming methodologies.

Currently the most efficient methodologies in linear programming are divided into two classes, namely the simplex-type methods and the interior point methods. The former consists in variants of the pioneering method in the area, the (primal) simplex method (Dantzig, 1951). The latter is given by methods which are motivated by nonlinear programming techniques, following the projective interior point method (Karmarkar, 1984). Since their first proposals, these two methods have been continuously improved. It has been a very active research area, not only due to theoretical developments, but also because of the search for techniques that results in efficient implementations of the methods. The main motivation lies in the fact that many other areas benefit from improvements in the linear programming methodologies, as mentioned in the previous paragraph. As a result of this intense research, simplex type-methods and interior point methods are very competitive nowadays. It is difficult to guess which method will solve a given linear programming problem with the best performance (Bixby, 2002; Gondzio, 2012). The dual simplex method is recognized for its best overall performance in relation to other simplex-type methods (Koberstein, 2008). Regarding interior point methods, the primal-dual interior point method is the most efficient variant in general (Gondzio, 2012).

In this chapter, we briefly review the linear programming methodologies, as they are addressed in the remaining chapters of this thesis. The purpose is to provide a self-contained description of the methodologies as well as to set a standard notation. In addition, we propose a unified framework to describe simplex type methods and interior point methods. Usually, these two methodologies are described by using very different notations, as if they were not related at all. In spite of being essentially different, these methods have certain similarities and, hence, they can be represented by a unified linear programming framework. In Section 2.2, we present the main ideas of simplex-type methods and then describe two important variants, the primal simplex method and the dual simplex method. In Section 2.3, we describe

the primal-dual interior point method and briefly discuss how to warmstart this method. We assume the reader is familiar with the linear programming subject. For a thorough introduction to linear programming we suggest the textbooks by Bertsimas and Tsitsiklis (1997); Arenales et al. (2007).

2.1 A unified framework for linear programming

Consider a linear programming problem which is formulated in the following *standard form* (P), together with its associated dual formulation (D),

$$\begin{array}{llll} (P) & \min & c^T x & \\ & \text{s.t.} & Ax = b & \\ & & x \geq 0, & \end{array} \qquad \begin{array}{llll} (D) & \max & b^T y & \\ & \text{s.t.} & A^T y + s = c & \\ & & s \geq 0, & \end{array} \tag{2.1}$$

where A is an $m \times n$ matrix with $\text{rank}(A) = m$, $0 < m \leq n$, $x \in \mathbb{R}^n$ is the vector of primal variables, $y \in \mathbb{R}^m$ and $s \in \mathbb{R}^n$ are the vectors of dual variables. The parameters $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are the costs vector and the right-hand side vector, respectively. We represent the columns of A by the vectors $a_j \in \mathbb{R}^m$, $j = 1, \ldots, n$. Formulation (P) is referred to as the *primal*, to distinguish it from the dual (D). The linear function $c^T x$ is called the *primal objective function*, while $b^T y$ is the *dual objective function*.

Any linear programming can be formulated in the standard form (2.1). Due to this simplicity, this formulation is typically adopted in textbooks and introductory texts regarding linear programming. Nevertheless, other types of formulations can be used to represent a problem equivalently. These alternative formulations are beneficial to special contexts and they may avoid the use of additional variables and/or constraints. For instance, in the literature about computational implementations of simplex type methods, a bounded variable formulation is preferred because it express explicitly the lower and upper bounds of the primal variables. Thus, it helps to describe the implementation issues in a more realistic way. In practice, a linear programming problem may require lower and upper bounds to be imposed to certain constraints. In such case, the *general form* can be used to formulate the problem, without having to add any slack nor surplus variables. As we show in Chapter 3, this formulation offers advantageous features which can be exploited in order to obtain a variant of the dual simplex method.

Let $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}$ be the primal feasible set, *i.e.*, the set of all the points that satisfies the constraints of (P). If $\mathcal{P} = \varnothing$, then the primal problem is *infeasible*. Otherwise, there exists at least one feasible point x in $\mathcal{P}$. In addition, if $c^T x \to -\infty$, *i.e.*, for any point $x \in \mathcal{P}$ it is possible to obtain another point $\bar{x} \in \mathcal{P}$ such that $c^T \bar{x} < c^T x$, then the primal problem is *unbounded*. Otherwise, if there exists a point $x^\star \in \mathcal{P}$ such that $c^T x^\star \leq c^T x$, for all $x \in \mathcal{P}$, then it is called an *optimal solution* of (P). Analogously, we can define the dual feasible set as $\mathcal{D} = \{(y, s) \in \mathbb{R}^{m+n} \mid A^T y + s = c,\ s \geq 0\}$. The dual problem is infeasible in case $\mathcal{D} = \varnothing$, and it is unbounded if the dual objective function goes to infinity. If $\mathcal{D} \neq \varnothing$ and the dual objective function is bounded above, there exists an optimal solution $(y^\star, s^\star)$ of the dual problem. In this context, we have an important result in linear programming: given a primal solution $x \in \mathcal{P}$ and a dual solution $(y, s) \in \mathcal{D}$, then $b^T y \leq c^T x$. In addition, for a pair of optimal primal and dual solutions, we have $b^T y = c^T x$. This result has also useful consequences. For instance, if the dual problem is unbounded, then the primal problem can only be infeasible.

Associated to the primal-dual pair of problems (2.1) we have the following first order *optimality conditions*, also known as the Karush-Kuhn-Tucker (KKT) conditions:

$$b - Ax = 0 \tag{2.2a}$$

$$c - A^T y - s = 0 \quad (2.2b)$$
$$XSe = 0 \quad (2.2c)$$
$$x \geq 0 \quad (2.2d)$$
$$s \geq 0 \quad (2.2e)$$

where $X = diag(x_1, ..., x_n)$, $S = diag(s_1, ..., s_n)$, and $e = (1, 1, ..., 1)^T$ is an n-vector. See (Nocedal and Wright, 2006, chap. 12) for a full description of how to obtain the optimality conditions (2.2). Both classes of linear programming methods work by relaxing one or more subset of the above conditions. Then, they iteratively modify a starting point until all these conditions are satisfied. Most simplex-type methods relax some of the nonnegativity conditions (2.2d) and (2.2e), but equations (2.2a)-(2.2c) must be satisfied at each iteration. To satisfy the complementarity slackness (2.2c), the variables are split into two sets. We define a basic set $\mathcal{B}$ and non-basic set $\mathcal{N}$. All non-basic primal variables are set to zero, *i.e.*, $x_j := 0$ for all $j \in \mathcal{N}$. Also, we set $s_i := 0$ for all $i \in \mathcal{B}$ and, hence, the equations in (2.2c) are fully satisfied. In the primal simplex method, the inequalities (2.2d) are always satisfied, while (2.2e) are satisfied only when an optimal solution is found. In the dual simplex method, the opposite is required. In the primal-dual variations of the simplex method, both (2.2d) and (2.2e) may be violated throughout the iterations. By using a different strategy, the primal-dual interior point method replace (2.2c) by $XSe = \mu e$, where $\mu > 0$ is the *barrier parameter*. This parameter is smoothly driven to zero so that (2.2c) is satisfied at the end of the iterations. The remaining equations (2.2a) and (2.2b) must be satisfied at each iteration in the (feasible) primal-dual interior point method, while this is not required for the infeasible variant of this method.

Both methodologies are iterative methods which start with an initial point (x^0, y^0, s^0) and generate a sequence of points (x^k, y^k, s^k) until either they reach an optimal solution $(x^\star, y^\star, s^\star)$ or they identify the problem has no optimal solution (because it is unbounded or infeasible). At a given iteration k, the current iterate (x^k, y^k, s^k) is modified by using the vector of *search directions* $(\Delta x^k, \Delta y^k, \Delta s^k)$. In addition, the methods adopt primal and dual *step-sizes* ε^P and ε^D, so that the new iterate is given by

$$(x^{k+1}, y^{k+1}, s^{k+1}) := (x^k, y^k, s^k) + (\varepsilon^P \Delta x^k, \varepsilon^D \Delta y^k, \varepsilon^D \Delta s^k). \quad (2.3)$$

Simplex type methods obtain the search direction by solving relatively simple linear systems which are determined by the basic matrix (the submatrix of A composed by the columns with indices in the basic set $\mathcal{B}$). The basic matrix is an $m \times m$ matrix which is efficiently represented in the implementations so that the linear systems can be solved by means of linear transformations. In addition, the representation of the basic matrix is typically updated at each iteration, which speed up the implementations. In interior point methods the search direction is obtained by solving the Newton-step equations, as they correspond to the direction provided by the Newton method. In practice, they may be computed by means of a symmetric indefinite augmented system as well as by using a semidefinite normal equations system. The former is described by an $(m + n) \times (m + n)$ matrix, while the matrix is $n \times n$ in the latter. Notice that these are larger matrices in relation to the basic matrix of simplex type methods (in practice, they may be considerably larger, as $n \gg m$ in most cases). As a consequence, a single iteration of an interior point method is in general significantly more expensive than that of simplex type methods. On the other hand, much less iterations are typically required by interior point methods.

Having briefly addressed the basic differences between simplex type methods and interior point methods, we now further describe the main variants of these methods in the next sections. In these descriptions, we follow several important publications regarding these subjects

(e.g. Maros, 2003a; Koberstein, 2005; Wright, 1997; Gondzio, 2012). In addition to that, we propose a unified framework to describe both methods. Usually, simplex type methods and interior point methods are described by using very different notations, as if they were not related at all. As discussed in the previous paragraph, we can state these two classes of methods by following a unified description as in (2.3). The particularities of each method are then specified when describing a given variant. In fact, the algorithms described in the next sections, are specialized from the linear programming framework presented in Algorithm 1.

Algorithm 1: Linear programming framework.

Input: matrix A; parameters c and b; initial solution (x, y, s); IT_MAX.
Output: optimal solution $x^\star$; or it detects the problem is infeasible or unbounded; or IT = IT_MAX.

```
1  IT = 0;
2  While (IT < IT_MAX) do
3  {
4      If the optimality conditions (2.2) are satisfied then STOP, an optimal solution has been found;
5      Compute the search direction (Δx, Δy, Δs) and the step-sizes ε^P and ε^D;
6      If ε^P → ∞ then STOP, the problem is unbounded;
7      If ε^D → ∞ then STOP, the problem is infeasible;
8      Update the current solution: (x, y, s) := (x, y, s) + (ε^P Δx, ε^D Δy, ε^D Δs);
9  }
```

2.2 Simplex type methods

Simplex type methods are based on the following geometric observation: if there is an optimal solution then there is an optimal vertex of the feasible region $\mathcal{P}$. If the optimal solution is unique, then it can only be a vertex of the feasible region. In case of multiple solutions, any point in the optimal facet is an optimal solution, including the vertices of $\mathcal{P}$ on it. By observing this, simplex type methods restrict their search to the vertices of the feasible region, without loss of generality. They start with an initial vertex and then iteratively moves from one vertex to an improved neighboring vertex until an optimal vertex is reached (see Fig. 2.1 for an illustration). Relying only on vertices of the feasible region is important because a vertex has a special feature: it can be represented by a vector with at most m nonzero components. Hence, the computations in a simplex-type method split the coefficient matrix by using a *basic set*. According to this set, m columns are selected to compose a basic matrix and only the variables associated to those columns may assume nonzero values. All the other variables are set as zero. The basic set simplifies the computations that are required at each iteration. This results in the relatively low computational cost to perform a single iteration of a simplex type method. In this section, we describe the main ideas used in the primal and the dual simplex methods for problems in the standard form (2.1). First, we formally define the concept of basic solutions.

2.2.1 Basic solutions

In the description of problem (2.1) we have assumed that $\text{rank}(A) = m$. It allows us to select a linearly independent set of m columns of the coefficient matrix A, which determines a *basis*. Let $\mathcal{B}$ denote the set of *basic indices* corresponding to the m columns in the basis. The nonsingular submatrix $B = A_{\mathcal{B}}$ which is composed by the columns with indices in $\mathcal{B}$ is called the *basic matrix*. It induces a partition of the coefficient matrix that is given by $A = [B \mid N]$, considering an implicit permutation of columns if needed (without loss of generality). N is the

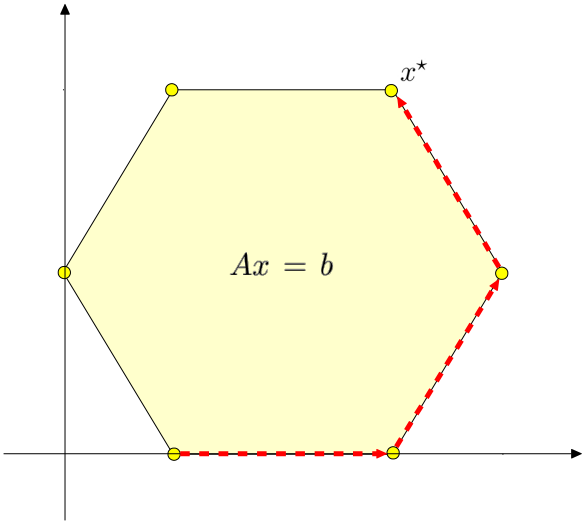

Figure 2.1: Illustration of the points that are visited by a simplex-type method. The feasible set is represented by the colored region. The vertices of the region are highlighted by the (yellow) balls on its boundary. A simplex type method visits a sequence of vertices, following the (red) dashed arrows (for example).

nonbasic matrix and the indices in $\mathcal{N}$ are called the nonbasic indices. This notation induces a partition in the vectors of the problem, such as $c = (c_\mathcal{B}, c_\mathcal{N})$ and $x = (x_\mathcal{B}, x_\mathcal{N})$. If $i \in \mathcal{B}$, then the variable x_i is basic (or, is in the basis). Otherwise, the variable is nonbasic (or, is not in the basis).

By using the basic partition $A = [B \mid N]$, we can rewrite the system of constraints $Ax = b$ as $Bx_\mathcal{B} + Nx_\mathcal{N} = b$. Since B is nonsingular, we express $x_\mathcal{B}$ in terms of the nonbasic variables. This leads to the *general solution*

$$x_\mathcal{B} = B^{-1}\left(b - Nx_\mathcal{N}\right). \tag{2.4}$$

Particular solutions can be obtained by setting values to $x_\mathcal{N}$. Here, we are interested in the particular solution that is stated in Definition 2.2.1.

Definition 2.2.1 (Primal basic solution)**.** *Consider a basic partition $A = [B \mid N]$ of the coefficient matrix of the primal problem (P) in (2.1). Let x be a particular primal solution which is determined from the general solution (2.4) by setting $x_\mathcal{N} = 0$, so that $x_\mathcal{B} = B^{-1}b$. x is called the* primal basic solution.

According to Definition 2.2.1, any primal basic solution satisfies the constraints $Ax = b$. In addition, its nonbasic components $x_\mathcal{N}$ further satisfy the nonnegativity constraints. Hence, in case we have $x_\mathcal{B} \geq 0$ the primal basic solution is feasible. Otherwise, we have an infeasible primal basic solution. If the basic components of a primal basic solution are all different from zero, then we say this solution is nondegenerate. Otherwise, the primal basic solution is said to be degenerate. This nomenclature is extended to the corresponding basis.

Consider now the dual problem (D) in (2.1). If we apply the basic partition $A = [B \mid N]$ to the coefficient matrix of its system of constraints, we obtain

$$\begin{bmatrix} B^T y \\ N^T y \end{bmatrix} + \begin{bmatrix} s_\mathcal{B} \\ s_\mathcal{N} \end{bmatrix} = \begin{bmatrix} c_\mathcal{B} \\ c_\mathcal{N} \end{bmatrix}. \tag{2.5}$$

The general solution of this system of equations is given by

$$\begin{cases} y^T &= c_{\mathcal{B}}^T B^{-1} - s_{\mathcal{B}}^T B^{-1}, \\ s_{\mathcal{N}}^T &= c_{\mathcal{N}}^T - y^T N. \end{cases} \tag{2.6}$$

In this context, a particular solution of great importance is stated in Definition 2.2.2.

Definition 2.2.2 (Dual basic solution)**.** *Consider a basic partition $A = [B \mid N]$ of the coefficient matrix of the primal problem (P) in (2.1). Let (y, s) be a particular dual solution which is determined from the general solution (2.6) by setting $s_{\mathcal{B}} = 0$, so that $y^T = c_{\mathcal{B}}^T B^{-1}$ and $s_{\mathcal{N}}^T = c_{\mathcal{N}}^T - y^T N$. (y, s) is called the* dual basic solution.

In case $s_{\mathcal{N}} \geq 0$, then the dual basic solution (y, s) is feasible. If this inequality holds strictly then the dual basic solution is nondegenerate. Otherwise, at least one component of $s_{\mathcal{N}}$ is equal to zero and we have a degenerate dual basic solution. In such case, we say that the basis is dual degenerate. The nonbasic components of the dual basic solution, $s_{\mathcal{N}}$ are often referred to as the *relative costs*, or the *reduced costs*.

Simplex type methods work with basic solutions only. Notice that the primal and dual basic solutions satisfy the complementarity conditions (2.2c). Indeed, according to Definitions 2.2.1 and 2.2.2, $s_{\mathcal{B}} = 0$ and $x_{\mathcal{N}} = 0$ result in $x_i s_i = 0$ and $x_j s_j = 0$, for $i \in \mathcal{B}$, $j \in \mathcal{N}$. Furthermore, an important result can be obtained by relating the basic solutions with the optimality conditions (2.2). If the primal and the dual basic solutions are both feasible, then the corresponding basis is optimal. This is the fundamental result for a simplex type method. Indeed, the method starts with a basic solution which satisfies conditions (2.2a) to (2.2c), but violates at least one of the set of conditions (2.2d) or (2.2e), depending on the variant. Then, by performing a sequence of basis changes iteratively, the method reaches a basic solution that also satisfies conditions (2.2d) and (2.2e) and, hence, this is an optimal solution of the problem. In the next section, we describe the primal simplex method for problems in the standard form (2.1).

2.2.2 Primal simplex method

Consider a linear programming problem which is formulated as (P) in (2.1). Assume that we know a basic partition $A = [B \mid N]$ of the coefficient matrix of (P), such that the corresponding primal basic solution x is feasible. Suppose that the corresponding dual solution (y, s) is not feasible and, hence, x cannot be an optimal solution of (P). In order to improve the current solution, we perform a *basis change*. To obtain the new basis we follow the *(primal) simplex strategy*, which consists in perturbing a nonbasic component of the primal solution, as we describe below. At each iteration of the primal simplex method, we perform a basis change by using this strategy. The method terminates if both primal and dual basic solutions are feasible, or if it detects the problem is unbounded.

Assume that the r-th nonbasic component of the dual basic solution is infeasible, *i.e.*, $s_{\mathcal{N}_r} < 0$. We perturb the primal component $x_{\mathcal{N}_r}$ by using a perturbation ε^P, so that it becomes

$$\overline{x}_{\mathcal{N}_r} = x_{\mathcal{N}_r} + \varepsilon^P = \varepsilon^P. \tag{2.7}$$

All the remaining nonbasic components remain equal to zero. On the other hand, the basic components are affected by this perturbation. Indeed, by using the general solution (2.4), we obtain

$$\overline{x}_{\mathcal{B}} = B^{-1}(b - \varepsilon^P a_{\mathcal{N}_r}) = x_{\mathcal{B}} - \varepsilon^P B^{-1} a_{\mathcal{N}_r}. \tag{2.8}$$

We can represent the results of perturbing the basic and nonbasic components in a uniform way. Let Δx be a vector defined as

$$\Delta x = \begin{bmatrix} \Delta x_{\mathcal{B}} \\ \Delta x_{\mathcal{N}} \end{bmatrix} := \begin{bmatrix} -B^{-1}a_{\mathcal{N}_r} \\ e_r \end{bmatrix}, \tag{2.9}$$

where e_r is the r-th column of the identity matrix of order $n-m$. We call Δx as the primal search direction. By using this notation, the new primal solution which we obtain after the perturbation is given by

$$\overline{x} = x + \varepsilon^P \Delta x. \tag{2.10}$$

From (2.10), we see that the perturbation ε^P is actually the primal step-size. It determines how far we go along direction Δx. Notice that the value of the objective function corresponding to the new primal solution is

$$\begin{aligned} c^T\overline{x} &= c^T x + \varepsilon^P c^T \Delta x \\ &= c^T x - \varepsilon^P c_{\mathcal{B}}^T B^{-1} a_{\mathcal{N}_r} + \varepsilon^P c_{\mathcal{N}}^T e_r \\ &= c^T x + \varepsilon^P (c_{\mathcal{N}_r} - y^T a_{\mathcal{N}_r}) \\ &= c^T x + \varepsilon^P s_{\mathcal{N}_r} \\ &\leq c^T x, \end{aligned} \tag{2.11}$$

where (y, s) is the dual basic solution with $s_{\mathcal{N}_r} < 0$. Therefore, Δx is a descending direction for the primal solution x. The reduction in the value of the objective function depends on the step-size ε^P, so we wish to make this value as large as possible. However, we must be careful to not make the primal solution infeasible after the perturbation. More specifically, we have to analyze each component in (2.10) in order to preserve $x + \varepsilon^P \Delta x \geq 0$. The nonbasic components $x_{\mathcal{N}}$ are perturbed by $\varepsilon^P e_r \geq 0$, so they do not violate this requirement. The same happens with the basic components in which $\Delta x_{\mathcal{B}_i} \geq 0$. On the other hand, for the components in which $\Delta x_{\mathcal{B}_i} < 0$, we have that

$$x_{\mathcal{B}_i} + \varepsilon^P \Delta x_{\mathcal{B}_i} \geq 0 \;\Leftrightarrow\; \varepsilon^P \leq -x_{\mathcal{B}_i}/\Delta x_{\mathcal{B}_i}, \tag{2.12}$$

Hence, to obtain the largest ε^P we have to perform the *ratio test*. First, we compute the values

$$\varepsilon_i^P = \begin{cases} -x_{\mathcal{B}_i}/\Delta x_{\mathcal{B}_i}, & \text{if } \Delta x_{\mathcal{B}_i} < 0, \\ +\infty, & \text{otherwise.} \end{cases} \tag{2.13}$$

for each $i = 1, \ldots, m$. Then, the primal step-size is computed as $\varepsilon^P = \min_{i=1,\ldots,m} \varepsilon_i^P$. In case $\varepsilon^P = +\infty$, then the problem is unbounded, as the value of the objective function can be made as large as we may want along the direction Δx. Otherwise, the step-size ε^P is determined by at least one basic index. Assume this happen for the p-th basic index, so that $\varepsilon^P = -x_{\mathcal{B}_p}/\Delta x_{\mathcal{B}_p}$. Observe that for this component, we have $\overline{x}_{\mathcal{B}_p} = x_{\mathcal{B}_p} + (-x_{\mathcal{B}_p}/\Delta x_{\mathcal{B}_p})\Delta x_{\mathcal{B}_p} = 0$. This suggests a new basic partition, in which the p-th basic component becomes nonbasic. On the other hand, the r-th nonbasic component must become basic, as it may be positive after the perturbation. Therefore, we perform a basis change by swapping indices $\mathcal{B}_p$ and $\mathcal{N}_r$. If $x_{\mathcal{B}} > 0$, then $\varepsilon^P > 0$ and hence the value of the objective function is strictly improved after the basis change – see (2.11).

The perturbation of the primal components induces a perturbation of the dual components as well. To be in accordance with the basis change, the component $s_{\mathcal{N}_r}$ must become equal to zero, while $s_{\mathcal{B}_p}$ is allowed to be nonnegative. Hence, we perturb $s_{\mathcal{B}}$ by using the dual step-size $\varepsilon^D \geq 0$, such that after the perturbation it becomes $\overline{s}_{\mathcal{B}} = s_{\mathcal{B}} + \varepsilon^D e_p = \varepsilon^D e_p$,

where e_p is the p-th column of the identity matrix of order m. Notice that only the p-th basic component is perturbed. According to the dual general solution (2.6), this perturbation affects the components of y and the nonbasic components $s_{\mathcal{N}}$. Indeed, after adding the perturbation we obtain

$$\begin{aligned} \overline{y}^T &= c_{\mathcal{B}}^T B^{-1} - \varepsilon^D e_p^T B^{-1} \\ &= y^T + \varepsilon^D \Delta y^T, \end{aligned}$$

where $\Delta y := -(e_p^T B^{-1})^T = -(B^{-1})^T e_p$. By using this result, the perturbation of the nonbasic dual components leads to

$$\begin{aligned} \overline{s}_{\mathcal{N}}^T &= c_{\mathcal{N}}^T - (y^T + \varepsilon^D \Delta y^T) N \\ &= s_{\mathcal{N}}^T - \varepsilon^D \Delta y^T N \\ &= s_{\mathcal{N}}^T + \varepsilon^D \Delta s_{\mathcal{N}}^T, \end{aligned} \tag{2.14}$$

where $\Delta s_{\mathcal{N}} := -(\Delta y^T N)^T = -N^T \Delta y$. If we further define $\Delta s_{\mathcal{B}} := e_p$, then the new dual solution we obtain after adding the perturbation is given by

$$(\overline{y}, \overline{s}) = (y, s) + \varepsilon^D (\Delta y, \Delta s). \tag{2.15}$$

The dual step-size ε^D must be chosen in order to satisfy $\overline{s}_{\mathcal{N}_r} = 0$, as the index $\mathcal{N}_r$ was chosen to enter the basis. Hence, we have that

$$\overline{s}_{\mathcal{N}_r} = 0 \;\Leftrightarrow\; s_{\mathcal{N}_r} + \varepsilon^D \Delta s_{\mathcal{N}_r} = 0 \;\Leftrightarrow\; \varepsilon^D = -s_{\mathcal{N}_r} / \Delta s_{\mathcal{N}_r}.$$

This ratio is well-defined and leads to $\varepsilon^D > 0$, as we have $s_{\mathcal{N}_r} < 0$ and $\Delta s_{\mathcal{N}_r} > 0$. Indeed $s_{\mathcal{N}_r}$ is the infeasible component of the nonbasic dual solution which motivates the perturbation of the primal solution – see (2.7). In addition, we have that

$$\begin{aligned} \Delta s_{\mathcal{N}_r} &= \Delta s_{\mathcal{N}}^T e_r \\ &= -\Delta y^T N e_r \\ &= e_p^T B^{-1} N e_r \\ &= e_p^T B^{-1} a_{\mathcal{N}_r} \\ &= -e_p^T \Delta x_{\mathcal{B}} \\ &= -\Delta x_{\mathcal{B}_p}. \end{aligned} \tag{2.16}$$

From the ratio test operation, we have $\Delta x_{\mathcal{B}_p} < 0$ and thus $\Delta s_{\mathcal{N}_r} > 0$. Therefore, even when the primal step-size ε^P is equal to zero, then the dual step-size ε^D is strictly larger than zero.

In summary, we have defined the search direction $(\Delta x, \Delta y, \Delta s)$ and the step-sizes ε^P and ε^D which will be used to obtain new primal and dual solutions. The new solutions suggest a basis change, in which the current r-th nonbasic index becomes basic, and the current p-th basic index becomes nonbasic. If $\varepsilon^P > 0$, then the value of the objective function is strictly reduced after the basis change. Otherwise, the value remains the same (but it never becomes worse). The basis change that we described determines a single iteration of the primal simplex method.

Algorithm 2 presents the primal simplex method for problems in the standard form. The parameter IT_MAX sets the maximum number of iterations. In line 7, the algorithm checks the optimality conditions (2.2). Only the nonbasic components of the dual solution s can violate the optimality conditions. In case they are feasible, then the current primal basic

solution is optimal and the algorithm terminates. Otherwise, the *pricing operation* selects an infeasible component $s_{\mathcal{N}_r}$. If more than one component is infeasible, then we follow a given rule to select one of them. Several pricing rules are proposed in the literature. For instance, in the Dantzig's rule we select the index which corresponds to the minimum $s_{\mathcal{N}_i} < 0$ (ties can be broken arbitrarily). See Maros (2003a) for a review of the pricing rules that are the most used in practice.

Algorithm 2: Primal simplex method for problems in the standard form.

Input: matrix A; parameters c and b; basic partition $A = [B \mid N]$ which is primal feasible; IT_MAX.
Output: optimal solution x; or it detects the problem is unbounded; or IT reaches IT_MAX.

IT = 0;
Compute the primal basic solution: $x_{\mathcal{B}} := B^{-1}b$ and $x_{\mathcal{N}} := 0$;
Compute the dual basic solution: $y := c_{\mathcal{B}}^T B^{-1}$, $s_{\mathcal{B}} := 0$ and $s_{\mathcal{N}} := c_{\mathcal{N}} - N^T y$;

While (IT < IT_MAX) do
{
    If ($s_{\mathcal{N}} \geq 0$) then STOP, *an optimal solution has been found*;
    Pricing operation: choose an index r such that $s_{\mathcal{N}_r} < 0$;
    Compute the primal search direction: $\Delta x_{\mathcal{B}} := -B^{-1}a_{\mathcal{N}_r}$ and $\Delta x_{\mathcal{N}} := e_r$;

    Set $p := -1$;
    Ratio test: $p := \arg\min_{i=1,\ldots,n}\{-x_{\mathcal{B}_i}/\Delta x_{\mathcal{B}_i} \mid \Delta x_{\mathcal{B}_i} < 0\}$;
    If ($p = -1$) then STOP, *the problem is unbounded*;
    Compute the primal step-size: $\varepsilon^P = -x_{\mathcal{B}_p}/\Delta x_{\mathcal{B}_p}$;

    Compute the dual search direction: $\Delta y := -(B^{-1})^T e_p$, $\Delta s_{\mathcal{B}} := e_p$ and $\Delta s_{\mathcal{N}} := -N^T \Delta y$;
    Compute the dual step-size: $\varepsilon^D = -s_{\mathcal{N}_r}/\Delta s_{\mathcal{N}_r}$;

    Update the current solution: $(x, y, s) := (x, y, s) + (\varepsilon^P \Delta x, \varepsilon^D \Delta y, \varepsilon^D \Delta s)$;
    Basis change: $\mathcal{N}_r$ becomes the p-th basic index and $\mathcal{B}_p$ becomes the r-th nonbasic index;
    IT = IT + 1;
}

Notice that Algorithm 2 requires as input a partition which is primal feasible. In practice, this information may be difficult to known in advance. In such case, a phase-1 algorithm may be called before Algorithm 2 in order to obtain a basic feasible solution (Maros, 1986). Alternatively, we can recur to the big-M strategy, which inserts artificial variables to the problem (Bertsimas and Tsitsiklis, 1997). It is worth mentioning that in order to have a successful implementation of Algorithm 2, it is important to rely on appropriate computational techniques. For a review on the main techniques we suggest the publications by Maros (2003a) and Munari (2009).

2.2.3 Dual simplex method

Given the linear programming problem (P) in (2.1), consider a basic partition $A = [B \mid N]$ of the coefficient matrix, such that the corresponding dual basic solution (y, s) is feasible. Assume that the primal basic solution x is not feasible and, hence, it cannot be optimal. We propose to obtain a new basic solution by using the dual simplex strategy, which consists in perturbing a basic component of the dual solution. Since the perturbed component may become positive, the resulting dual solution suggests a basis change.

Assume that the primal basic solution x has a basic component $x_{\mathcal{B}_p} < 0$ and hence it is infeasible. We propose to perturb the corresponding dual component by making $\overline{s}_{\mathcal{B}_p} =$

$s_{\mathcal{B}_p} + \varepsilon^D = \varepsilon^D$, where ε^D is a given perturbation. The remaining basic components are kept at zero, so that we have

$$\overline{s}_{\mathcal{B}} = s_{\mathcal{B}} + \varepsilon^D \Delta s_{\mathcal{B}}, \tag{2.17}$$

where $\Delta s_{\mathcal{B}} := e_p$ and e_p is the p-*th* column of the identity matrix of order m. From the general solution (2.6), we see that this perturbation results in modifying the components of the dual solution y as follows

$$\begin{aligned} \overline{y}^T &= c_{\mathcal{B}}^T B^{-1} - \varepsilon^D e_p^T B^{-1} \\ &= y^T + \varepsilon^D \Delta y^T, \end{aligned}$$

where $\Delta y := -(B^{-1})^T e_p$, exactly as we have defined in the previous section. Furthermore, the nonbasic components become

$$\begin{aligned} \overline{s}_{\mathcal{N}}^T &= c_{\mathcal{N}}^T - (y + \varepsilon^D \Delta y)^T N \\ &= s_{\mathcal{N}}^T - \varepsilon^D \Delta y^T N \\ &= s_{\mathcal{N}}^T + \varepsilon^D \Delta s_{\mathcal{N}}^T, \end{aligned} \tag{2.18}$$

where $\Delta s_{\mathcal{N}} := -N^T \Delta y$, which follows the same definition as in the previous section as well. Hence, the perturbation of the dual solution is given by $(\overline{y}, \overline{s}) = (y, s) + \varepsilon^D (\Delta y, \Delta s)$,. From this expression, we see that ε^D is the step-size we adopt along the dual direction $(\Delta y, \Delta s)$. Notice that the value of the objective function which corresponds to the perturbed dual solution is given by

$$\begin{aligned} b^T \overline{y} &= b^T (y + \varepsilon^D \Delta y) \\ &= b^T y - \varepsilon^D b^T (B^{-1})^T e_p \\ &= b^T y - \varepsilon^D (B^{-1} b)^T e_p \\ &= b^T y - \varepsilon^D x_{\mathcal{B}_p}. \end{aligned}$$

Therefore, we see that the objective function changes by the amount $-\varepsilon^D x_{\mathcal{B}_p} > 0$ in relation to its previous value. Since $x_{\mathcal{B}_p}$ is strictly smaller than zero, a step-size $\varepsilon^D > 0$ results in strictly improving the value of the objective function. The change in this value is proportional to ε^D, so we should set this step-size as large as possible. On the other hand, we want to keep the dual solution feasible after the perturbation. In particular, we must guarantee that $\overline{s}_{\mathcal{N}} \geq 0$. The other dual components remain feasible, as $s_{\mathcal{B}_p}$ is the only basic component we perturb, and y is a free variable. Observe in (2.18) that only the negative components of $\Delta s_{\mathcal{N}}$ may cause $\overline{s}_{\mathcal{N}}$ to become infeasible. In particular for such components, we must guarantee that $s_{\mathcal{N}_i} + \varepsilon^D \Delta s_{\mathcal{N}_i} \geq 0$, which leads to

$$\varepsilon^D \leq -s_{\mathcal{N}_i} / \Delta s_{\mathcal{N}_i}. \tag{2.19}$$

This suggests a ratio test in which we compute the ratio in (2.19) for each nonbasic index and, then, we set ε^D as the smallest ratio. This leads to the largest step-size such that the new dual solution $(\overline{y}, \overline{s})$ is feasible. Formally, we compute $\varepsilon^D = \min_{i=1,\ldots,n-m} = \varepsilon_i^D$, where

$$\varepsilon_i^D = \begin{cases} -s_{\mathcal{N}_i} / \Delta s_{\mathcal{N}_i}, & \text{if } \Delta s_{\mathcal{N}_i} < 0, \\ +\infty, & \text{otherwise.} \end{cases} \tag{2.20}$$

If we obtain $\varepsilon^D = +\infty$ after performing the ratio test, then the dual problem (D) is unbounded and, thus, the primal problem (P) is infeasible. Otherwise, we have $\varepsilon^D = \varepsilon_r^D$ for at least one

index r, $r = 1, \ldots, n-m$ (in case of ties, we can break them arbitrarily). Notice that by using this step-size we obtain

$$\overline{s}_{\mathcal{N}_r} = s_{\mathcal{N}_r} + \varepsilon^D \Delta s_{\mathcal{N}_r} = s_{\mathcal{N}_r} + \left(-\frac{s_{\mathcal{N}_r}}{\Delta s_{\mathcal{N}_r}} \right) \Delta s_{\mathcal{N}_r} = 0.$$

On the other hand, recall that the p-th basic component becomes $\overline{s}_{\mathcal{B}_p} = \varepsilon^D = -s_{\mathcal{N}_r}/\Delta s_{\mathcal{N}_r}$. This suggests a basis change, in which the p-th basic index becomes nonbasic and the r-th nonbasic index becomes basic. Associated to this new basis, we have the basic dual solution $(\overline{y}, \overline{s}) = (y, s) + \varepsilon^D(\Delta y, \Delta s)$, where the dual step-size ε^D and the dual search direction $(\Delta y, \Delta s)$ were defined above.

The primal solution x must be perturbed as well in order to reflect the basis change. By using the perturbation $\varepsilon^P \geq 0$, we perturb the r-th nonbasic component only, *i.e.*,

$$\overline{x}_{\mathcal{N}} = x_{\mathcal{N}} + \varepsilon^P e_r = \varepsilon^P \Delta x_{\mathcal{N}},$$

where $\Delta x_{\mathcal{N}} := e_r$ and e_r is the unitary vector which corresponds to the r-th column of the identity matrix of order $n-m$. According the primal general solution (2.4), this perturbation modifies the basic components as follows

$$\overline{x}_{\mathcal{B}} = B^{-1}b - \varepsilon^P B^{-1} N e_r = x_{\mathcal{B}} + \varepsilon^P \Delta x_{\mathcal{B}},$$

where $\Delta x_{\mathcal{B}} := -B^{-1}Ne_r = -B^{-1}a_{\mathcal{N}_r}$. Hence, $\Delta x = (\Delta x_{\mathcal{B}}, \Delta x_{\mathcal{N}})$ is the primal search direction, as defined in (2.9). In addition, the perturbation ε^P corresponds to the primal step-size that we take along direction Δx. To compute ε^P, we take into account the assumption that the index $\mathcal{B}_p$ becomes nonbasic after the basis change. Hence, we must have $\overline{x}_{\mathcal{B}_p} = 0$, which implies in

$$\varepsilon^P = -\frac{x_{\mathcal{B}_p}}{\Delta x_{\mathcal{B}_p}}.$$

This ratio is well-defined, since $\Delta x_{\mathcal{B}_p} = -\Delta s_{\mathcal{N}_r} > 0$ (see (2.16)). In addition, we have $\varepsilon^P > 0$ as $x_{\mathcal{B}_p} < 0$ is the infeasible primal component which motivates the perturbation in (2.17).

The discussion presented so far describes the key ideas of the dual simplex method. This method starts with a basis which is dual feasible, but primal infeasible. At each iteration, the method checks if the current basic solution is optimal by verifying the feasibility of the primal solution. If the primal solution is also feasible, then it is an optimum of problem (P) in (2.1), as the dual solution is feasible at any iteration. Otherwise, a basis change is performed in order to obtain a new basis. Algorithm 3 presents the dual simplex method for problems in the standard form. It is very similar to the primal simplex method given in Algorithm 2. The main difference is in the choice of the variables that will be swapped in the basis change. Indeed, the primal method select first the index that will enter the basis, and then determines which index will leave the basis. On the other hand, in the dual method the index that will leave the basis is selected first.

In Algorithm 3, we assume that a basic partition with a feasible dual basic solution is given as an input. In case a basic partition with such feature is not known in advance, a dual phase-1 method should be called before calling Algorithm 3. Different types of dual phase-1 methods are proposed in the literature (Kostina, 2002; Koberstein and Suhl, 2007). Alternatively, we can use the big-M approach, although it typically causes the numerical instability of the simplex method (Bertsimas and Tsitsiklis, 1997; Koberstein and Suhl, 2007). An efficient and stable implementation of Algorithm 3 requires the use of appropriate computational techniques, which is out of the scope of this chapter. For the reader interested in these techniques we suggest the publications by Maros (2003a) and Koberstein (2005).

Algorithm 3: Dual simplex method for problems in the standard form.

Input: matrix A; parameters c and b; basic partition $A = [B \mid N]$ which is dual feasible; IT_MAX.
Output: optimal solution x; or it detects the problem is infeasible; or IT reaches IT_MAX.

IT = 0;
Compute the primal basic solution: $x_{\mathcal{B}} := B^{-1}b$ and $x_{\mathcal{N}} := 0$;
Compute the dual basic solution: $y := c_{\mathcal{B}}^T B^{-1}$, $s_{\mathcal{B}} := 0$ and $s_{\mathcal{N}} := c_{\mathcal{N}} - N^T y$;

While (IT < IT_MAX) do
{
    If ($x_{\mathcal{B}} \geq 0$) then STOP, *an optimal solution has been found*;
    Pricing operation: choose an index p such that $x_{\mathcal{B}_p} < 0$;
    Compute the dual search direction: $\Delta y := -(B^{-1})^T e_p$, $\Delta s_{\mathcal{B}} := e_p$ and $\Delta s_{\mathcal{N}} := -N^T \Delta y$;

    Set $p := -1$;
    Ratio test: $r := \arg\min_{i=1,\ldots,n-m}\{-s_{\mathcal{N}_i}/\Delta s_{\mathcal{N}_i} \mid \Delta s_{\mathcal{N}_i} < 0\}$;
    If ($p = -1$) then STOP, *the problem is infeasible*;
    Compute the dual step-size: $\varepsilon^D = -s_{\mathcal{N}_r}/\Delta s_{\mathcal{N}_r}$;

    Compute the primal search direction: $\Delta x_{\mathcal{B}} := -B^{-1}a_{\mathcal{N}_r}$ and $\Delta x_{\mathcal{N}} := e_r$;
    Compute the primal step-size: $\varepsilon^P = -x_{\mathcal{B}_p}/\Delta x_{\mathcal{B}_p}$;

    Update the current solution: $(x, y, s) := (x, y, s) + (\varepsilon^P \Delta x, \varepsilon^D \Delta y, \varepsilon^D \Delta s)$;
    Basis change: $\mathcal{N}_r$ becomes the p-th basic index and $\mathcal{B}_p$ becomes the r-th nonbasic index;
    IT = IT + 1;
}

2.3 The primal-dual interior point method

In this section, we describe the primal-dual interior point method, which is currently recognized as the most efficient variant of interior point methods for linear programming (Gondzio, 2012). This method is based on the perturbation of the optimality conditions (2.2), in which we replace the complementarity conditions (2.2c) by $XSe = \mu e$, where $\mu > 0$ is the *barrier parameter*, or *duality measure*. As a result, we obtain the *perturbed* optimality conditions

$$b - Ax = 0 \tag{2.21a}$$
$$c - A^T y - s = 0 \tag{2.21b}$$
$$XSe = \mu e \tag{2.21c}$$
$$x \geq 0 \tag{2.21d}$$
$$s \geq 0 \tag{2.21e}$$

Theoretically, this is the main difference of the primal-dual interior point method in relation to simplex type methods. Instead of keeping the pairwise products $x_i s_i$ always equal to zero, the primal-dual interior point method allow them to be equal to $\mu > 0$. This parameter is typically loose at the very first iterations, but then it is gradually driven to zero throughout the iterations. Hence, μ converges smoothly to an optimal solution which satisfies the complementarity conditions (2.2c).

Since we perturb the complementarity conditions, the iterates of the primal-dual interior point method are points in the interior of the positive orthant. For a given value of μ, the system (2.21) has a unique solution, which is called a μ-center. The set composed of all μ-centers is called a *central-path*. Fig. 2.2 illustrates these concepts in the space of pairwise products (with n=2). The positive orthant is the colored region in the figure. The (red)

dashed lines represent the set of points which satisfy the complementarity conditions (2.2c). The continuous line in the middle of the positive orthant represents the central path. For a given a value $\mu > 0$, the μ-center is a point in the central path that satisfies $x_1 s_1 = x_2 s_2 = \mu$.

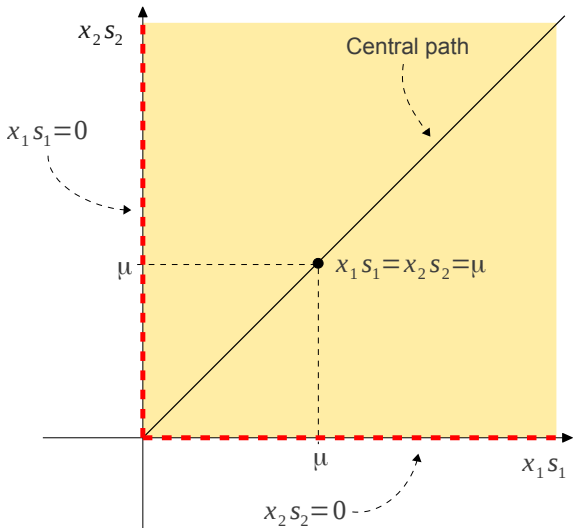

Figure 2.2: Illustration of the positive orthant in the space of pairwise products, with $n = 2$.

Instead of strictly satisfying the perturbed optimality conditions, the iterates of the primal-dual interior point method belong to a neighborhood of the central path. The idea of the neighborhood is to keep the iterates well-centered and in a safe area so that all variables approach their optimal values with a uniform pace. Different neighborhoods have been proposed in the literature, and they differ by the way they measure the distance of the pairwise products $x_i s_i$ to the duality measure μ. For instance, the *narrow neighborhood* is defined as

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

where $\mathcal{F}^0 = \{(x, y, s) \mid Ax = b, A^T y + s = c \text{ and } (x, s) > 0\}$ is the set of positive primal-dual feasible solutions, and $\theta \in [0, 1)$ is a given parameter. The set of pairwise products that satisfy the inequality imposed by $\mathcal{N}_2(\theta)$ is illustrated in Fig. 2.3 for iterates k and $k + p$, for any given $k > 0$ and $p > 0$. The continuous line inside the positive orthant represents the central path. The (black) points in the central path are the μ-centers corresponding to the duality measures μ^k and μ^{k+p}. We can define a neighborhood by using the ∞-norm as well. By adopting only the lower bound on $x_i s_i$, we obtain the *wide neighbourhood*, which is defined as

$$\mathcal{N}_{-\infty}(\gamma) = \{(x, y, s) \in \mathcal{F}^0 \mid x_i s_i \geq \gamma\mu, \forall i = 1, \ldots, n\},$$

where $\gamma \in (0, 1)$ is a fixed parameter. Notice that this neighborhood is less restrictive than $\mathcal{N}_2(\theta)$. An illustration of the $\mathcal{N}_{-\infty}(\gamma)$ in the set of pairwise products is given in Fig. 2.4. In the wide neighborhood, if we further impose upper bounds on the pairwise products, then we obtain the *symmetric neighborhood*, which is define as

$$\mathcal{N}_s(\gamma) = \{(x, y, s) \in \mathcal{F}^0 \mid \gamma\mu \leq x_i s_i \leq \frac{1}{\gamma}\mu, \forall i = 1, \ldots, n\},$$

with $\gamma \in (0, 1)$. This neighborhood is illustrated in Fig. 2.5. The choice of the neighborhood affects the theoretical as well as practical properties of the algorithm. If the primal-dual

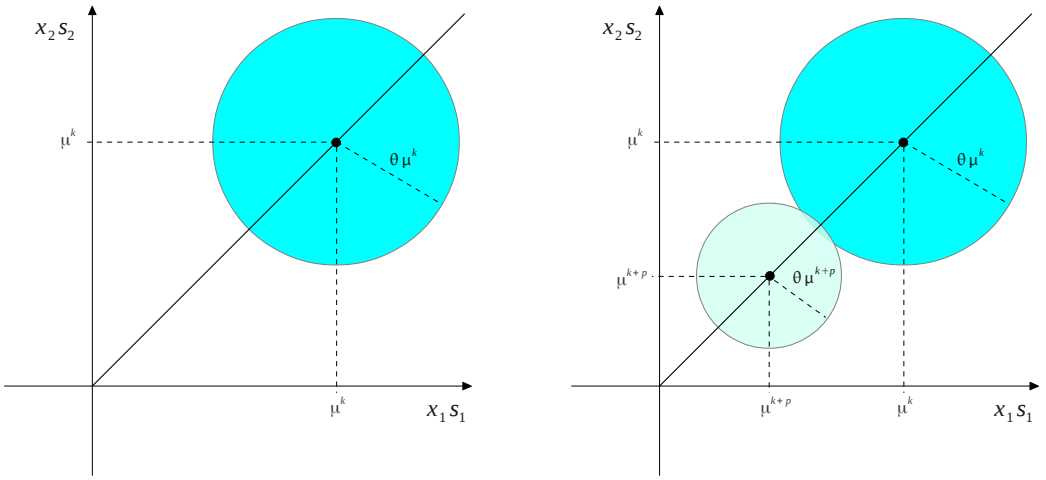

Figure 2.3: Narrow neighborhood in the space of pairwise products with the duality measures of iterations k and $k+p$ (the set $\mathcal{F}^0$ is not represented in the illustration).

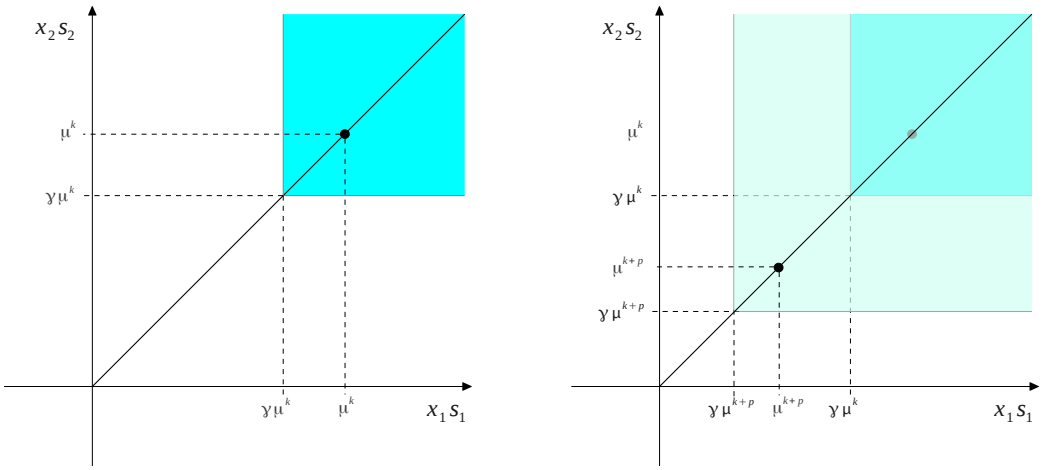

Figure 2.4: Wide neighborhood in the space of pairwise products with the duality measures of iterations k and $k+p$ (the set $\mathcal{F}^0$ is not represented in the illustration).

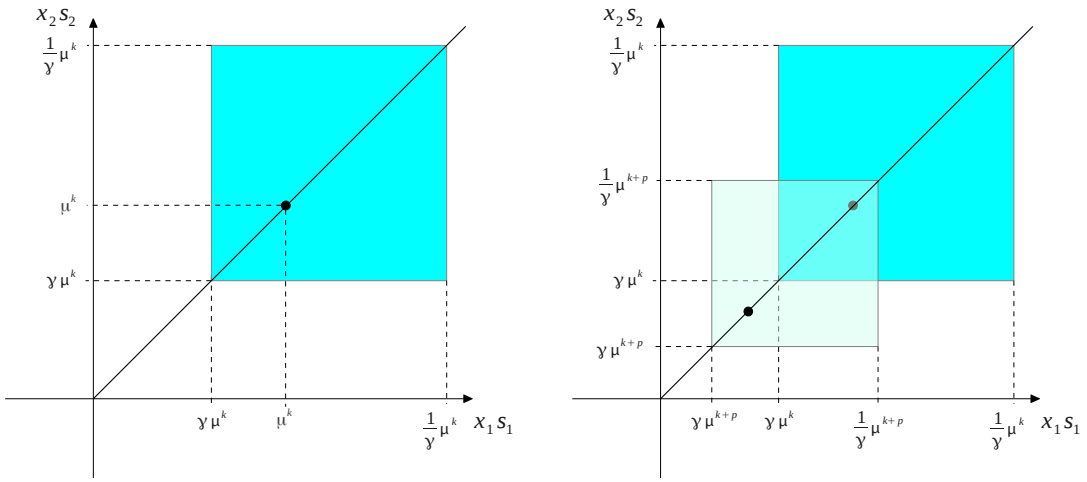

Figure 2.5: Symmetric neighborhood in the space of pairwise products with the duality measures of iterations k and $k+p$ (the set $\mathcal{F}^0$ is not represented in the illustration).

interior point algorithm is based on the narrow neighborhood $\mathcal{N}_2$, then it achieves a solution that satisfies $\mu < \epsilon$ in $O(\sqrt{n}|\log(1/\epsilon)|)$ iterations, by starting from a solution that satisfies $\mu \leq 1/\epsilon^{\kappa}$, for some positive κ. By using the wide neighborhood $\mathcal{N}_{-\infty}$, the worst-case complexity of the method is a bit worse, given by $O(n|\log(1/\epsilon)|)$ iterations. In spite of this, the practical performance of the method that uses the $\mathcal{N}_{-\infty}$ is typically superior to the method using the $\mathcal{N}_2$. The theoretical and practical features of the symmetric neighborhood is similar to that of the wide neighborhood, but it is preferred by some authors (Colombo and Gondzio, 2008; Gondzio, 2012).

The description of the primal-dual interior point algorithm follows the linear programming framework defined by Algorithm 1. At each iteration, given an iterate (x, y, s), we compute the primal-dual search direction $(\Delta x, \Delta y, \Delta s)$ and the primal and dual step-sizes, ε^P and ε^D, so that the next iterate is given by

$$(\overline{x}, \overline{y}, \overline{s}) = (x, y, s) + (\varepsilon^P \Delta x, \varepsilon^D \Delta y, \varepsilon^D \Delta s).$$

The search direction $(\Delta x, \Delta y, \Delta s)$ is obtained by solving the Newton step equations

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e - XSe \end{bmatrix}, \tag{2.22}$$

where $\sigma \in [0, 1]$ is a parameter used to reduce the complementarity gap $x^T s$ of the next iterate. For $\sigma = 0$, the system of equations (2.22) corresponds to a first order approximation of the perturbed optimality conditions (2.21). We obtain this approximation by applying one step of the Newton method. The solution with $\sigma = 0$ is the affine-scaling direction, which corresponds to the usual direction of the Newton method. On the other hand, $\sigma = 1$ determines a centering direction toward a μ-center in the central path, which has all the pairwise products identical to μ. Hence, by setting an intermediate value for σ we obtain a combination of these two extreme directions. The resulting direction contributes to both optimality and centrality. After obtaining the primal-dual search direction, the step-sizes ε^P and ε^D are computed by performing a line search along the direction $(\Delta x, \Delta y, \Delta s)$. The chosen step-sizes must guarantee that the next iterate belongs to the used neighborhood.

Following the discussion that we have presented so far, the primal-dual interior point method is stated in Algorithm 4. In the algorithm, we denote by $\mathcal{N}_\star$ a neighborhood of the central path, which can be any of those described above. In line 7, the algorithm obtains the primal-dual search direction by solving the Newton step equations (2.22). In practice, we reformulate this system of equations in order to obtain a more compact coefficient matrix which is in addition symmetric. The idea is to obtain a more efficient representation of the matrix, which contributes to a better overall performance of the method. Two types of reformulations are used in practice, depending on the structure of the coefficient matrix of (2.22). They are known as the *augmented system* and the *normal equations*. Recall that x and s are positive at any iteration of the primal-dual interior point algorithm, so the matrices X and S are nonsingular. The augmented system consists in eliminating Δs from step equations, so that we obtain

$$\begin{aligned} \begin{bmatrix} 0 & A \\ A^T & -D^{-2} \end{bmatrix} \begin{bmatrix} \Delta y \\ \Delta x \end{bmatrix} &= \begin{bmatrix} 0 \\ s - \sigma\mu X^{-1} e \end{bmatrix}, \\ \Delta s &= -s + \sigma\mu X^{-1} e - X^{-1} S \Delta x, \end{aligned} \tag{2.23}$$

where $D := S^{-\frac{1}{2}} X^{\frac{1}{2}}$. In the normal equations, we go a step further and eliminate Δx from (2.23) and, hence, we obtain

$$AD^2 A^T \Delta y = A(x - \sigma\mu S^{-1} e), \tag{2.24}$$

$$\begin{aligned} \Delta s &= -A^T \Delta y, \\ \Delta x &= -x + \sigma\mu S^{-1} e - S^{-1} X \Delta s. \end{aligned}$$

The normal equations are widely used in practice. The coefficient matrix in (2.24) is symmetric positive semidefinite. Hence, this matrix is represented by the Cholesky factorization, which is very efficient in practice. It requires appropriate techniques to avoid numerical instabilities caused by small components of x and s, which typically appears in the very last iterations. The coefficient matrix in (2.23) is symmetric indefinite, which makes its factorization more complicated than in the normal equations. Nevertheless, the augmented system is more efficient when the matrix A has one or more dense columns, which justifies its use.

Algorithm 4: Primal-dual interior point method for problems in the standard form.

Input: matrix A; parameters c and b; barrier reduction parameter $\sigma \in (0,1)$; IT_MAX; primal-dual feasible solution $(x, y, s) \in \mathcal{N}_\star$.
Output: optimal solution $x^\star$; or IT = IT_MAX.

```
1  IT = 0;
2  Set the barrier parameter: μ := (x^T s)/n;
3  While (IT < IT_MAX) do
4  {
5      If the optimality conditions (2.2) are satisfied then STOP, an optimal solution has been found;
6      Reduce the barrier parameter: μ := σμ;
7      Compute the primal-dual search direction (Δx, Δy, Δs) by solving the Newton step equations;
8
9      Find ε^P and ε^D as the largest step-sizes such that (x + ε^P Δx, y + ε^D Δy, s + ε^D Δs) ∈ N_⋆;
10     Reset the step-sizes: ε^P := 0.99ε^P and ε^D := 0.99ε^D;
11
12     Update the solution: (x, y, s) := (x, y, s) + (ε^P Δx, ε^D Δy, ε^D Δs);
13 }
```

Algorithm 4 requires the initial solution to belong to a given neighborhood. Hence, the initial solution must be primal and dual feasible, so the problem cannot be infeasible neither unbounded. In practice, a solution with such feature may be difficult to obtain. To overcome this, we can recur to the *infeasible* primal-dual interior point method. This algorithm is very similar to Algorithm 4, but it relies in a relaxed neighborhood of the central path. This neighborhood does not require the iterates to satisfy the systems of constraints $Ax = b$ and $A^T y + s = c$, but only the positivity conditions $(x, s) > 0$. The method deals with the infeasibilities by incorporating them to the right-hand side of the Newton step equations (2.22), *i.e.*,

$$\begin{bmatrix} A & 0 & 0 \\ 0 & A^T & I \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} b - Ax \\ c - A^T y - s \\ \sigma\mu e - XSe \end{bmatrix}. \tag{2.25}$$

The modified neighborhood simplifies the line search which determines the primal and dual step-sizes. Indeed, ε^P and ε^D are taken as the largest values such that $x + \varepsilon^P \Delta x > 0$ and $s + \varepsilon^D \Delta s > 0$. Therefore, the step-sizes are computed by using a ratio test, similar to that used in simplex-type methods (see Algorithms 2 and 3). For further details on the infeasible primal-dual interior point algorithm, we suggest the paper by Gondzio (2012).

2.3.1 Warmstarting the primal-dual interior point algorithm

In this section, we address the reoptimization of a problem by using the primal-dual interior point method. The reoptimization is important to situations in which closely-related problems

are solved in sequence. By closely-related problems we mean that the problems differ by changes such as inserting/dropping constraints and/or columns, fixing/relaxing variables, and perturbing the problem data. These situations arise, for example, in the context of the column generation method and of the branch-and-bound method. By using a warmstarting technique, we exploit some available information in order to define an advanced initial point to start the primal-dual interior point method. The goal is to improve the performance of solving the problem in relation to starting from a standard initial point. In the next paragraphs, we briefly describe the issues involved in the warmstarting operation. The purpose is to present the main ideas which are proposed in the literature. The theoretical and computational details of the warmstarting strategies is out of the scope of this chapter. They can be found in the references presented throughout the discussion.

It is well-known the ability of simplex-type methods to reoptimize similar problems from a previous optimal solution. Typically, by starting from the optimal basis of the previous problem, a relatively small number of basis changes are enough to reach the optimal solution of the new problem. In addition, the optimal basis of the previous problem is usually either primal feasible or dual feasible to the new problem. Nevertheless, for interior point methods, the optimal solution of a closely-related problem is not a good initial point in general. Indeed, this solution is likely to be at or very close to the boundary of the primal-dual feasible set $\mathcal{F}^0$. In such case, the initial point is badly centered, so it leads to many iterations with very small step-sizes, which slow down the convergence of the method. In addition to be badly centered, the initial point may violate some primal or dual constraints as well, which is an issue for the feasible primal-dual interior point method. Therefore, the reoptimization in interior point methods cannot be interpreted very literally as in simplex-type methods.

There are two main concerns regarding the warmstarting point in interior point methods. First, it should satisfy the primal and dual constraints. In case of the infeasible variant of the method, the infeasibility of the initial point should be small enough to be absorbed in the first iterations of the method. Second, the pairwise products must be positive and should not deviate too much from each other. In other words, the iterates must be well-centered in the positive orthant. These requirements are illustrated in Fig. 2.6, which shows different types of candidate initial points. Suitable initial points are represented by the (green) balls in the figure. They are feasible as shown in part (a), and well-centered in the positive orthant, as shown in part (b). For the infeasible primal-dual interior point method, the point represented by a (yellow) triangle in part (a) should be suitable as well. The (red) squares in the figure correspond to infeasible and/or badly centered points, so they are not suitable as initial points.

In the last 15 years, the research regarding efficient warmstarting techniques for interior point methods has been very active. We classify these techniques into two groups: (*i*) one-phase techniques; and (*ii*) two-phase techniques. In the one-phase techniques, the problem is modified by the addition of slack variables and/or penalization factors so that the optimal solution (x, y, s) of a closely-related problem is used as the initial point. The problem is modified in such a way that (x, y, s) is not too close to the boundary of the modified feasible set. Fig. 2.7 illustrates the idea of the one-phase techniques. Recall that in Fig. 2.6, several candidate initial points are available, but some of them were not good initial points in the space of pairwise products. In the one-phase techniques we expand this space, by relaxing the nonnegativity constraints, as illustrated in Fig. 2.7. As a result, all the points become acceptable initial points and can be used as a warmstarting point. Of course, the expanded boundaries must be driven to zero again during the solution of the problem. Warmstarting techniques in this group have been proposed by Benson and Shanno (2007) and Engau et al. (2009, 2010). They differ from each other by how the space of pairwise products is expanded.

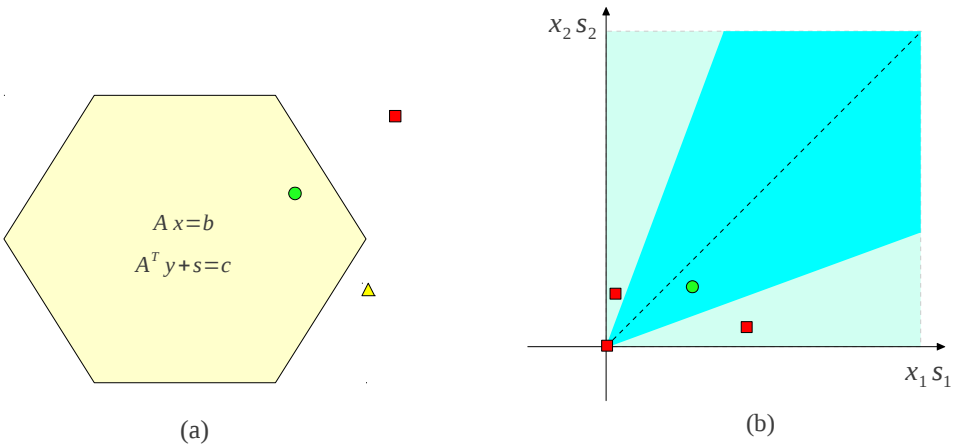

Figure 2.6: The two main concerns when warmstarting the primal-dual interior point method: (a) Feasibility of the primal and dual constraints; (b) Nonnegativity and centrality.

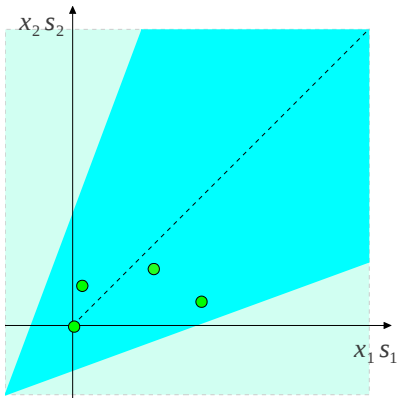

Figure 2.7: One-phase warmstarting techniques in the space of pairwise products. The positive orthant is expanded so that all points represented in Fig. 2.6 become suitable initial points.

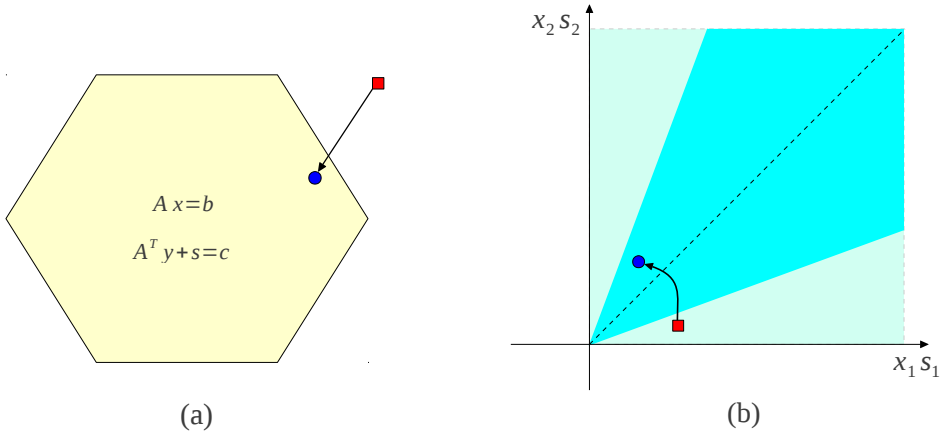

Figure 2.8: Two-phase warmstarting techniques in the space of pairwise products. The vector of adjustments is computed so that the bad points represented in Fig. 2.6 become suitable initial points after using the adjustments.

Two-phase techniques are characterized by modifying one or more stored points of a closely-related problem. In the first phase, a stored point (x, y, s) is adjusted by a vector $(\Delta x, \Delta y, \Delta s)$ so that it becomes feasible (or slightly infeasible) and better centered in the positive orthant. The adjustment can be performed in a single step or by using an iterative process in which several stored points are modified through one or more adjustments. In the second phase, the adjusted point $(x+\Delta x, y+\Delta y, s+\Delta s)$ is used as the initial point. Fig. 2.8 illustrates the idea of the two-phase techniques. The arrows represent the adjustments which are applied to the stored points. The adjusted points are represented by a (blue) ball in the figure. The first two-phase warmstarting techniques for the primal-dual interior point method were proposed by Mitchell and Borchers (1996) and Gondzio (1998). Those papers have proposed the basic ideas which are used in the subsequent techniques proposed by Yildirim and Wright (2002); Gondzio and Grothey (2003, 2008); John and Yildirim (2008).

To further clarify the main ideas of the warmstarting techniques, we close this section by briefly describing the strategy proposed by Gondzio (1998). This strategy is the basis of the warmstarting procedures that are implemented in the interior point solver HOPDM (Gondzio, 1995, 2012). This solver is used in the computational experiments presented in Chapters 4 and 5 of this thesis. Gondzio's warmstarting technique was proposed for the case in which new columns are added to the problem after solving it. Nevertheless, the key ideas used in this strategy have been successfully extended to other contexts Gondzio and Grothey (2003).

Consider a linear programming problem which is formulated as the primal problem (P) in (2.1). Assume that after solving this problem, we modify it by adding $\tilde{n} > 0$ columns to its coefficient matrix. We represent the new columns by a submatrix $\tilde{A}$. The variables corresponding to these columns are represented by the vector $w \in \mathbb{R}^{\tilde{n}}$. The changes in the primal problem result in new constraints to the dual problem (D). The resulting primal-dual pair of problems is given by

$$\begin{array}{llc} (\tilde{P}) & \min & c^T x + \tilde{c}^T w \\ & \text{s.t.} & Ax + \tilde{A}w = b \\ & & x, w \geq 0, \end{array} \qquad \begin{array}{llc} (\tilde{D}) & \max & b^T y \\ & \text{s.t.} & A^T y + s = c \\ & & \tilde{A}^T y + z = \tilde{c} \\ & & s, z \geq 0. \end{array} \tag{2.26}$$

We want to solve the primal problem $(\tilde{P})$ in (2.26) by using some information from the closely-related problem (P) in (2.1). Gondzio (1998) proposes a warmstarting technique which uses a close-to-optimality approximate μ-center which is stored during the solution of the closely-related problem. Since this point may violate the feasible set of the new problem $(\tilde{P})$, the author suggests to compute an adjustment in order to modify the stored point. The adjustment should generate a warmstarting point that is nearly feasible and relatively close to the central path. As observed by the author, since we do not know the new columns in advance, the choice of the μ-center is a safe option because it is the center of an ellipsoid inscribed in the dual feasible region. Hence, it is possible to take at least a small step from this point in order to correct the infeasibilities caused by the new columns.

To store the μ-center that will be used in a subsequent problem, the solution process need to be split into two stages. The interior point method iterates until the required tolerance ε_μ of the μ-center is achieved (*e.g.*, $\varepsilon_\mu = 0.001$). Then, if the current iterate is not sufficiently well-centered, a few additional centering steps are carried out in order to obtain a point satisfying

$$\gamma\mu \leq x_j s_j \leq (1/\gamma)\mu,$$

for some $\gamma > 0$ (e.g., $\gamma = 0.5$). The resulting point is stored to be used in a subsequent problem and the method continues iterating until the optimality tolerance is achieved. The

author recommends the use of centrality correctors (Gondzio, 1996; Colombo and Gondzio, 2008) to efficiently compute the centering steps.

Assume we have stored a μ-center (x, y, s) of the closely related problem (P) in (2.1). We want to use this point to define a warmstarting point $(\tilde{x}, \tilde{w}, \tilde{y}, \tilde{s}, \tilde{z})$ for solving the problem $(\tilde{P})$ in (2.26), with $(\tilde{x}, \tilde{w}) > 0$ and $(\tilde{s}, \tilde{z}) > 0$. Let $v = \tilde{c} - \tilde{A}^T y$ be the vector of $\tilde{n}$ components which corresponds to the depths of the new columns. We say that a column j is deep if $v_j < -\mu^{1/2}$. The components of the dual slack variable $\tilde{z}$ are set as $\tilde{z}_j = \max(\sqrt{\mu}, |v_j|)$. The choice $\tilde{w} = 0$ would lead to primal feasibility, but we need $\tilde{w} > 0$. By setting $\tilde{w} = \mu \tilde{Z}^{-1} e$ we have that the new pairwise products are equal to the former barrier parameter μ, *i.e.*, $\tilde{W}\tilde{Z}e = \mu e$. To obtain suitable values for the variables $\tilde{x}$, $\tilde{y}$ and $\tilde{s}$, we use the adjustment vector $(\Delta x, \Delta y, \Delta s)$. We compute the primal and the dual adjustments independently. The primal adjustment is given by

$$\Delta \bar{x} = -D^2 A^T (AD^2 A^T)^{-1} \tilde{A} \tilde{w}.$$

Then, we compute a step-size α^P such that $(x + \alpha^P \Delta x, \alpha^P \tilde{w})$ is feasible. For the dual variables, the adjustments are given by

$$\begin{aligned} \Delta y &= (AD^2 A^T)^{-1} \tilde{A} (\tilde{A}^T (AD^2 A^T)^{-1} \tilde{A})^{-1} v, \\ \Delta s &= -A^T \Delta y. \end{aligned}$$

A full step towards this direction in the dual space restores the feasibility of the new dual constraints. However, this step-size may not be possible as Δs depends on v, which may be large so s may become zero or negative. Thus, we compute the maximum step length α^D so that the dual variables are still positive after the adjustment. Therefore, the resulting point is given by

$$(\tilde{x}, \tilde{w}, \tilde{y}, \tilde{s}, \tilde{z}) = (x + \alpha^P \Delta x, \alpha^P \tilde{w}, y + \alpha^D \Delta y, s + \alpha^D \Delta s, \alpha^D \tilde{z}),$$

which is used as the warmstarting point to solve problem $(\tilde{P})$ in (2.26). This point can be slightly infeasible, but the remaining primal-dual infeasibilities will be removed during the optimization if we use the infeasible primal-dual interior point method.

2.4 Concluding remarks

In this chapter, we have described the main linear programming methodologies, namely the primal simplex method, the dual simplex method and the primal-dual interior point method. These methods are addressed in the remaining chapters of this thesis, and we follow the same notation and nomenclature that was stated here. We have presented the methods by using a unified linear programming framework. This allows us to see what are the differences as well as the similarities regarding these methodologies. In addition, we have presented the main issues regarding the warmstarting of the primal-dual interior point method, as the methodologies proposed in Chapters 4 and 5 depends on efficient warmstarting techniques to work well in practice.

Chapter 3

A dual simplex-type algorithm for problems in the general form

Simplex-type methods are widely used nowadays. This class of methods was recognized as one of the ten most influential algorithms on science and engineering in the 20th century (Dongarra and Sullivan, 2000). In addition, Elwes (2012) states that the simplex method is *"the algorithm that runs the world"*. According to the author, implementations of simplex-type methods are called upon thousands of times a second over the world, in order to solve problems related to our day-to-day lives. As a consequence, the search for more efficient variants of the simplex method has the potential to benefit many different areas.

In the last years, the researches regarding simplex-type methods have focused on different branches. For instance, the papers by Hall and McKinnon (2005), Koberstein (2008) and Hu and Pan (2008) are mainly concerned with efficient techniques that can be used to speed-up the computational implementations of the method. Maros (2003b), Paparrizos et al. (2003) and Pan (2008) propose new variants of simplex-type methods, which exploit different strategies for selecting the variables that enter/leave the basis. Recently, the community has raised again its attention to the theoretical complexity of the simplex method, due to a paper by Santos (2012) in which the author disprove the Hirsch conjecture (see e.g. Dantzig and Thapa, 1997). Further theoretical breakthroughs were presented by Friedmann (2011) and Friedmann et al. (2011). The authors prove the exponential complexity of two nonstandard pivot rules that behave very well in practice.

In this chapter, we address a variant of the dual simplex method, in which we exploit special features of a type of formulation. In particular, we exploit the *general form*, a formulation in which all the constraints are represented as inequalities with lower and upper bounds, including the nonnegativity of the decision variables. As it will be seen, this formulation leads to a piecewise linear objective function in the dual problem. We exploit this feature to modify the choice of the dual step-size, in order to obtain larger steps. Besides, the proposed modification helps to get rid of degenerate basic solutions.

It is worth mentioning that many solvers allow the user to enter with a linear programming problem in the general form. However, before calling the simplex method, they first transform the general form to the standard form presented in Chapter 2, by adding slack and/or surplus variables. What we propose in this chapter is the opposite strategy. The simplex method is designed in order to solve the problem in the general form, without the need to transform the formulation. The reason is that the general form can be exploited in order to improve the performance of the dual simplex method. In addition, we discard the need to add slack and surplus variables to the formulation. Another interesting feature is that we can insert

a new constraint $l_j \leq a^j x \leq u_j$ without changing the dimension of the basic matrix. This is important when the simplex method is used to solve a sequence of closely-related linear relaxations, a common scenario when it is applied within an integer programming methodology. For instance, a new constraint of this type may be generated after branching in the branch-and-bound method, or after adding a cut in the cutting plane method.

In the remainder of this chapter, we address the dual simplex method for problems in the general form. This study extends the earlier investigations regarding this method, which were presented by Arenales (1984), Sousa et al. (2005) and Silva et al. (2007). In particular, the main contributions are:

- We describe how to explicitly handle bounds that are equal to $-\infty$ or $+\infty$ (Section 3.1). Hence, we avoid the need for artificial bounds, which are known to cause numerical instability of the method.

- To have a successful implementation of the method, it is essential to use efficient techniques to represent and update the basic matrix. Hence, we describe how to extend the LU factorization and LU update schemes proposed by Suhl and Suhl (1990, 1993) to the particular case we are dealing with (Section 3.6.2). These techniques are known to be more efficient than those used in the computational implementations of Sousa et al. (2005) and Silva et al. (2007). They typically reduce the fill-in of the sparse representation of the basic matrix, and improve the numerical stability of the method.

- We show how to update the primal and dual solutions after carrying out a basis change, in order to avoid recomputing them from scratch at each iteration (Propositions 3.3.3 and 3.5.2).

- The performance of the method is verified by computational experiments using benchmarking instances from the public repository Netlib (Section 3.7). We compare the method against other simplex-type variants.

Apart from the importance of simplex-type methods and the active research regarding them, there is still a large gap between the techniques which are described in the literature and those which are implemented by commercial solvers. In other words, many theoretical and computational developments which are used to speed up the commercial implementations of the simplex-type methods are not reported in the literature. As a consequence, the authors are often discouraged in testing new strategies, as the computational performance of their implementations may be far from that of commercial solvers. With the investigation addressed in this paper, we hope to contribute with the literature by proposing a new theoretical and computational study regarding the dual simplex algorithm.

3.1 General form

Typically, we can model a linear programming problem by using different types of formulations. This freedom in representing a problem allows us to choose the formulation that most suits our needs. For instance, a given formulation may be easier to be understood in a textbook, while another leads to a better performance of the solver. Here, we consider that a linear programming problem is formulated in the following *general form*:

$$\min \quad f(x) = c^T x \tag{3.1a}$$

$$\text{s.t.} \quad l \leq Ax \leq u, \tag{3.1b}$$

where A is a real matrix $m \times n$ with $0 < n \leq m$ and $\text{rank}(A) = n$. The i-th column of A is denoted by the vector $a_i \in \mathbb{R}^m$, while the j-th row of this matrix is denoted by $a^j \in \mathbb{R}^n$, with $i \in \mathcal{I} := \{1, \ldots, n\}$ and $j \in \mathcal{J} := \{1, \ldots, m\}$. The vectors $c \in \mathbb{R}^n$, $l \in \mathbb{R}^m \cup \{-\infty\}$ and $u \in \mathbb{R}^m \cup \{+\infty\}$ are parameters of the problem, while $x \in \mathbb{R}^n$ is the vector of variables. The component c_i is the cost associated to variable x_i, for each $i \in \mathcal{I}$. For each $j \in \mathcal{J}$, a lower bound l_j and an upper bound u_j is defined for the j-th constraint. Notice that these bounds can be real values as well as $-\infty$ or $+\infty$. As a consequence, we are able to represent any type of linear constraint with this formulation. For instance, the equality $a^j x = b_j$ for a given $b_j \in \mathbb{R}$ can be represented by $l_j \leq a^j x \leq u_j$ with $l_j = u_j = b_j$. In addition, a nonnegative variable can be represented by the constraint $l_j \leq x_j \leq u_j$ with $l_j = 0$ and $u_j = +\infty$.

We classify the constraints in (3.1b) according to the respective components of bounds l and u. Consider the following partition of set $\mathcal{J}$:

$$\begin{aligned}
\mathcal{J}^b &= \{j \in \mathcal{J} \mid l_j \in \mathbb{R} \text{ and } u_j \in \mathbb{R}\}, \\
\mathcal{J}^l &= \{j \in \mathcal{J} \mid l_j \in \mathbb{R} \text{ and } u_j = +\infty\}, \\
\mathcal{J}^u &= \{j \in \mathcal{J} \mid l_j = -\infty \text{ and } u_j \in \mathbb{R}\}, \\
\mathcal{J}^f &= \{j \in \mathcal{J} \mid l_j = -\infty \text{ and } u_j = +\infty\}.
\end{aligned}$$

Since it is a partition, we have $\mathcal{J}^b \cup \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f = \mathcal{J}$ and $\mathcal{J}^b \cap \mathcal{J}^l \cap \mathcal{J}^u \cap \mathcal{J}^f = \varnothing$. A constraint in $\mathcal{J}^b$ is called *boxed*. If in addition $l_j = u_j$, then we have a *fixed* constraint. Constraints in sets $\mathcal{J}^l$ and $\mathcal{J}^u$ must have only one finite bound. Finally, *free* constraints belong to set $\mathcal{J}^f$. Strictly speaking, they are not constraints, but we extend the nomenclature in order to follow a standard presentation.

We can obtain a dual formulation for the general form (3.1). For the sake of clarity, we obtain this dual by using an equivalent formulation of (3.1), given by

$$\begin{aligned}
\min \quad & f(x) = c^T x && \text{(3.2a)} \\
\text{s.t.} \quad & Ax = w, && \text{(3.2b)} \\
& l \leq w \leq u. && \text{(3.2c)}
\end{aligned}$$

The vector w is used to represent the constraints in (3.1b). The nomenclature of each constraint according to the partition of $\mathcal{J}$ is extended to the corresponding component of w. From formulation (3.2), we define the associated *lagrangian problem*:

$$\begin{aligned}
h(y) &:= \min_{x,w} \{c^T x + y^T (w - Ax) \mid l \leq w \leq u\} && \text{(3.3a)} \\
&= \min_{x,w} \{(c^T - y^T A)x + y^T w \mid l \leq w \leq u\} && \text{(3.3b)} \\
&= \min_{x} (c^T - y^T A)x + \min_{w} \{y^T w \mid l \leq w \leq u\} && \text{(3.3c)}
\end{aligned}$$

where the components of $y \in \mathbb{R}^m$ are the Lagrange multipliers. For any y and any x, $h(y)$ gives a lower bound for $f(x)$. Indeed, we have

$$\begin{aligned}
f(x) &\geq \min_{x,w} \{c^T x \mid Ax = w, l \leq w \leq u\} \\
&= \min_{x,w} \{c^T x + y^T (w - Ax) \mid Ax = w, l \leq w \leq u\} \\
&\geq \min_{x,w} \{c^T x + y^T (w - Ax) \mid l \leq w \leq u\} \\
&= h(y)
\end{aligned}$$

Different values of y may lead to different lower bounds of $f(x)$ and, hence, we are interested in obtaining the best among them. The largest lower bound can be obtained by solving the (lagrangian) dual problem

$$\max h(y),\ y \in \mathbb{R}^m. \tag{3.4}$$

The representation of this problem can be improved by taking advantage of special cases in the definition of the lagrangian problem (3.3). In particular, we add new constraints to (3.3) in order to avoid $h(y) \to -\infty$. First, observe that x is free and thus for any $y \in \mathbb{R}$ we have

$$\min_x (c^T - y^T A)x \to -\infty, \tag{3.5}$$

whenever $c^T - y^T A \neq 0$. To avoid this, we restrict the domain of the Lagrange multipliers by imposing $c^T - y^T A = 0$. In other words, we additionally require $y \in \mathcal{D} := \{y \in \mathbb{R}^m \mid y^T A = c^T\}$ in (3.3). For the second term in expression (3.3c), notice that

$$\begin{aligned} \min_w \left\{ y^T w \mid l \leq w \leq u \right\} &= \min_w \left\{ \sum_{j \in \mathcal{J}} y_j w_j \mid l_j \leq w_j \leq u_j, j \in \mathcal{J} \right\} \\ &= \sum_{j \in \mathcal{J}} \min_{w_j} \left\{ y_j w_j \mid l_j \leq w_j \leq u_j \right\} \\ &= \sum_{\substack{j \in \mathcal{J} \\ y_j > 0}} y_j l_j + \sum_{\substack{j \in \mathcal{J} \\ y_j < 0}} y_j u_j, \end{aligned} \tag{3.6}$$

as the variables w_j are independent of each other. For each $j \in \mathcal{J}$, let $h_j(y_j)$ be a piecewise linear function which is defined as follows

$$h_j(y_j) := \begin{cases} l_j y_j, & \text{if } y_j > 0, \\ u_j y_j, & \text{if } y_j < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.7}$$

Using this definition together with (3.3), (3.6) and the definition of $\mathcal{D}$ given above, we obtain

$$h(y) = \sum_{j \in \mathcal{J}} h_j(y_j).$$

Since the components l_j and u_j may be $-\infty$ or ∞ , we may have $h_j(y_j)$ equal to $-\infty$ or $+\infty$ for a given $j \in \mathcal{J}$. To avoid this undesirable situation, we further restrict the domain of the Lagrange multipliers by requiring: $y_j \leq 0$, if $l_j = -\infty$ and $u_j \in \mathbb{R}$; $y_j \geq 0$, if $l_j \in \mathbb{R}$ and $u_j = \infty$; and $y_j = 0$, if $l_j = -\infty$ and $u_j = \infty$. In other words, we redefine $\mathcal{D}$ as

$$\mathcal{D} := \{y \in \mathbb{R}^m \mid y^T A = c^T, y_{\mathcal{J}^u} \leq 0, y_{\mathcal{J}^l} \geq 0, y_{\mathcal{J}^f} = 0\}.$$

If we incorporate all these changes to (3.4), the dual problem is given by

$$\max \quad h(y) = \sum_{j \in \mathcal{J}} h_j(y_j) \tag{3.8a}$$

$$\text{s.t.} \quad A^T y = c, \tag{3.8b}$$

$$y_j \leq 0,\ j \in \mathcal{J}^u, \tag{3.8c}$$

$$y_j \geq 0,\ j \in \mathcal{J}^l, \tag{3.8d}$$

$$y_j = 0,\ j \in \mathcal{J}^f. \tag{3.8e}$$

Observe that each $h_j(y_j)$ is a (concave) continuous piecewise linear function. If $l_j \neq u_j$, then it has a nondifferentiable point (breakpoint) at $y_j = 0$. Fig. 3.1 illustrates different plots that the function $h_j(y_j)$ may assume according to the bounds l_j and u_j. In part (a), $j \in \mathcal{J}^b$ and hence both bounds are finite. The slope of function $h_j(y_j)$ is $l_j > 0$ for $y_j > 0$, while it is given by $u_j > 0$ for $y_j < 0$. At $y_j = 0$, $h_j(y_j)$ has a breakpoint. The plot in part (b) is similar to (a), but with $l_j < 0$. In parts (c) and (d), only one of the bounds is finite and thus the domain of $h_j(y_j)$ is given by $\mathbb{R}^+$ in (c) and $\mathbb{R}^-$ in (d). The case $j \in \mathcal{J}^f$ is omitted, as it corresponds to the trivial situation in which $h_j(y_j)$ is defined for $y_j = 0$ only.

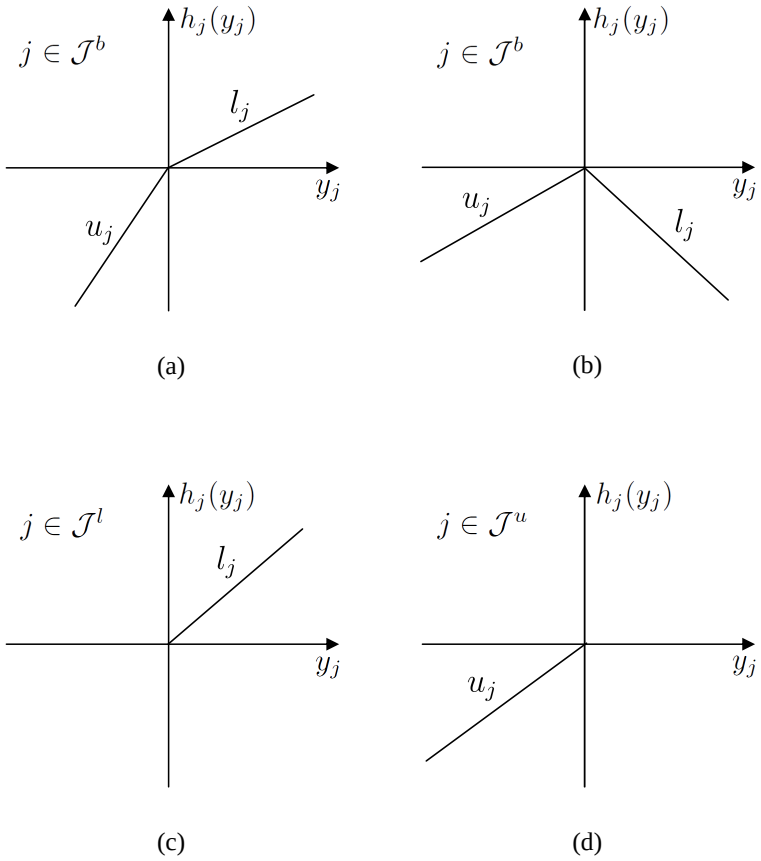

Figure 3.1: Different plots of the function $h_j(y_j)$, according to the bounds l_j and u_j, $j \in \mathcal{J}$.

Since $h(y)$ is given by the sum of each $h_j(y_j)$ over $j \in \mathcal{J}$, it is also a continuous piecewise linear function, which has breakpoints at $y_j = 0$, for all $j \in \mathcal{J}^b$ such that $l_j \neq u_j$. Dual objective functions with similar features also appear in standard formulations with bounded variables. In this context, these features have been successfully exploited to improve the performance of simplex-type methods (Fourer, 1994; Kostina, 2002; Maros, 2003b).

3.2 Basic solutions in the general form

Consider the constraints (3.8b) of the dual problem. Recall that we have assumed $\text{rank}(A) = n$ and hence we can select a linearly independent set of n columns of A^T in order to compose a *basis*. Let $\mathcal{B} \subset \mathcal{J}$ be an ordered set of n indices which correspond to the columns that we select to compose the basis. These indices are called the *basic indices*, while those in the ordered set $\mathcal{N} = \mathcal{J} \setminus \mathcal{B}$ are called the *nonbasic indices*.

The definition of a basis induces a *basic partition* of A^T which is given by $A^T = [B \mid N]$ (without loss of generality, an implicit permutation of the columns of A^T is considered in this representation). The *basic matrix* $B = A^T_{\mathcal{B}}$ is given by the columns of A^T which have indices in $\mathcal{B}$. Similarly, N is the *nonbasic matrix* which is composed by the columns with nonbasic indices. The basic partition is also extended to the vector of dual variables such that $y^T = \left(y^T_{\mathcal{B}}, y^T_{\mathcal{N}}\right)$. If $j \in \mathcal{B}$, we say that the variable y_j is basic (or, it is in the basis). Otherwise, we say y_j is nonbasic (or, it is not in the basis).

We can use a basic partition $A^T = [B \mid N]$ to rewrite constraints (3.8b) as

$$By_{\mathcal{B}} + Ny_{\mathcal{N}} = c.$$

Since B is nonsingular, we can rewrite the above expression in function of the nonbasic variables only. As a result, we obtain the *general solution* of the system $A^T y = c$, given by

$$y_{\mathcal{B}} = B^{-1}\left(c - Ny_{\mathcal{N}}\right). \tag{3.9}$$

Particular solutions may be obtained from (3.9) by setting values to $y_{\mathcal{N}}$. Here, we are interested in the particular solution that is given in Definition 3.2.1.

Definition 3.2.1 (Dual basic solution)**.** *Given a basic partition $A^T = [B \mid N]$ of the coefficient matrix of constraints (3.8b), let y be a particular dual solution which is determined from the general solution (3.9) by setting $y_{\mathcal{N}} = 0$, so that $y_{\mathcal{B}} = B^{-1}c$. y is called the* dual basic solution*.*

From Definition 3.2.1, we see that a dual basic solution must satisfy the constraints (3.8b). If in addition the variables $y_{\mathcal{B}_i}$ satisfy constraints (3.8c)-(3.8e) for each $i \in \{1, \ldots, n\}$ such that $\mathcal{B}_i \in \mathcal{J}^l \cup \mathcal{J}^u \cup \mathcal{J}^f$, then we have a *feasible* dual basic solution. Otherwise, the dual solution is *infeasible*. If $y_{\mathcal{B}_i} = 0$ for at least one index $i \in \{1, \ldots, n\}$, then the dual basic solution is *degenerate*. Otherwise, we have a nondegenerate dual basic solution. This nomenclature is extended to the associated basis.

Consider now the (primal) formulation (3.2). If we rewrite the set of constraints (3.2b) in terms of the basic partition $A^T = [B \mid N]$, then we obtain

$$\left\{ \begin{array}{rcl} B^T x & = & w_{\mathcal{B}} \\ N^T x & = & w_{\mathcal{N}} \end{array} \right. \Leftrightarrow \left\{ \begin{array}{rcl} x & = & \left(B^{-1}\right)^T w_{\mathcal{B}} \\ w_{\mathcal{N}} & = & N^T x \end{array} \right. , \tag{3.10}$$

which is the general solution of system $Ax = w$ in function of $w_{\mathcal{B}}$. For the sake of clarity, we further partition the set $\mathcal{B}$ into two disjoint subsets which are given by

$$\mathcal{B}^l = \{j \in \mathcal{B} \mid y_j \geq 0\} \quad \text{and} \quad \mathcal{B}^u = \{j \in \mathcal{B} \mid y_j < 0\}.$$

Following this notation, an important particular solution of (3.10) is given in Definition 3.2.2.

Definition 3.2.2 (Primal basic solution)**.** *Given a basic partition $A^T = [B \mid N]$ of the coefficient matrix of constraints (3.8b), let y be the associated dual basic solution as stated in Definition 3.2.1. The* primal basic solution *is the particular solution (x, w) which is obtained from (3.10) by fixing $w_{\mathcal{B}}$ as $w_{\mathcal{B}^l} = l_{\mathcal{B}^l}$ and $w_{\mathcal{B}^u} = u_{\mathcal{B}^u}$, so that $x = (B^{-1})^T w_{\mathcal{B}}$ and $w_{\mathcal{N}} = N^T x$.*

By definition, $Ax = w$ and $l_{\mathcal{B}} \leq w_{\mathcal{B}} \leq u_{\mathcal{B}}$ are both satisfied by a primal basic solution. However, the bounds of $w_{\mathcal{N}}$ may be violated, which leads to an *infeasible* primal basic solution. Otherwise, we have a *feasible* primal basic solution.

The basic solutions lead to a fundamental result: Given a basis, if the primal and the dual basic solutions are both feasible, then the basis is optimal. This result is formally stated in Proposition 3.2.3.

Proposition 3.2.3. *Given a basic partition $A^T = [B \mid N]$ of the coefficient matrix of problem (3.8), let (x, w) and y be the corresponding primal and dual basic solutions, respectively. If these basic solutions are feasible, then (x, w) is an optimal solution of (3.2) and y is an optimal solution of (3.8).*

Proof. As showed in Section 3.1, the dual objective function gives a lower bound of the optimal value of formulation (3.2), *i.e.*, $f(x) \geq h(y)$, for any x and y both feasible. Hence, it remains to prove that $f(x) = h(y)$. From Definitions 3.2.1 and 3.2.2 we have

$$f(x) \;=\; c^T x \;=\; c^T (B^{-1})^T w_{\mathcal{B}} \;=\; y_{\mathcal{B}}^T w_{\mathcal{B}} \;=\; y_{\mathcal{B}^l}^T w_{\mathcal{B}^l} + y_{\mathcal{B}^u}^T w_{\mathcal{B}^u}. \tag{3.11}$$

Since $y_{\mathcal{N}} = 0$, we obtain

$$y_{\mathcal{B}^l}^T w_{\mathcal{B}^l} + y_{\mathcal{B}^u}^T w_{\mathcal{B}^u} = \sum_{\substack{i=1 \\ y_i > 0}}^{m} y_i l_i + \sum_{\substack{i=1 \\ y_i < 0}}^{m} y_i u_i = h(y). \tag{3.12}$$

Therefore, $f(x) = h(y)$, which completes the proof. □

3.3 Improving a basic solution

Assume we have a basis such that the dual basic solution is feasible. In general, if the primal basic solution is infeasible, then we can make a *basis change* in order to improve the current value of the objective function. In a basis change, we select two indices in $\mathcal{J}$, one from $\mathcal{B}$ and another from $\mathcal{N}$, and swap them in order to obtain a new basis. If the dual basic solution is nondegenerate, then the value of dual objective function in the new basis is strictly greater than the value in the previous basis. In the degenerate case, the value may remain the same after the basis change (but it never becomes worse). Two situations can motivate a basis change. The first one is stated in Proposition 3.3.1.

Proposition 3.3.1. *Consider a basis with a feasible dual basic solution y. Suppose that the primal basic solution (x, w) is infeasible with $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$, for a given $s \in \{1, \ldots, m-n\}$. Let Δy be the* dual search direction *which is defined as*

$$\Delta y := \begin{bmatrix} \Delta y_{\mathcal{B}} \\ \Delta y_{\mathcal{N}} \end{bmatrix} = \begin{bmatrix} -B^{-1} a^{\mathcal{N}_s} \\ e_s \end{bmatrix}, \tag{3.13}$$

where $a^{\mathcal{N}_s}$ is the $\mathcal{N}_s$-th column of A^T and e_s is the s-th column of the identity matrix of order $m-n$. Then, we can obtain a new dual solution given by $\overline{y} = y + \varepsilon^D \Delta y$, for a given dual step-size *$\varepsilon^D \geq 0$, so that $\overline{y}$ is feasible and satisfies $h(\overline{y}) \geq h(y)$. In case the current basis is nondegenerate, then this inequality holds strictly.*

Proof. Suppose that we modify $y_{\mathcal{N}_s}$ by using the perturbation $\varepsilon^D \geq 0$, so that the new value of the nonbasic components are given

$$\overline{y}_{\mathcal{N}} = y_{\mathcal{N}} + \varepsilon^D e_s = \varepsilon^D e_s, \tag{3.14}$$

According to the general solution given by (3.9), this perturbation modifies the basic components as follows:

$$\begin{aligned} \overline{y}_{\mathcal{B}} &= B^{-1} \left(c - N \overline{y}_{\mathcal{N}} \right) \\ &= B^{-1} \left(c - \varepsilon^D N e_s \right) \end{aligned}$$

$$
\begin{aligned}
&= B^{-1}c - \varepsilon^D B^{-1}a^{\mathcal{N}_s} \\
&= y_{\mathcal{B}} + \varepsilon^D \left(-B^{-1}a^{\mathcal{N}_s}\right). \qquad (3.15)
\end{aligned}
$$

We can represent (3.14) and (3.15) uniformly by using the dual search direction defined in (3.13). Indeed, we have $\overline{y} = y + \varepsilon^D \Delta y$. In this expression, we see that ε^D is the step-size that determines how far the dual solution goes along the direction Δy. After the perturbation, the dual objective function is given by

$$
\begin{aligned}
h(\overline{y}) &= h(y + \varepsilon^D \Delta y) \\
&= \sum_{j=1}^{m} h_j(y_j + \varepsilon^D \Delta y_j) \\
&= \sum_{i=1}^{n} h_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D \Delta y_{\mathcal{B}_i}) + h_{\mathcal{N}_s}(\overline{y}_{\mathcal{N}_s}) \\
&= \sum_{i=1}^{n} h_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D \Delta y_{\mathcal{B}_i}) + \varepsilon^D l_{\mathcal{N}_s}, \qquad (3.16)
\end{aligned}
$$

as $\overline{y}_{\mathcal{N}_s} = \varepsilon^D$. Suppose that ε^D is chosen such that $\text{sign}(\overline{y}_j) = \text{sign}(y_j)$ for each $j \in \mathcal{B}$, *i.e.*, the sign of the components of the dual solution remain the same after the perturbation. Thus,

$$
\begin{aligned}
h(\overline{y}) &= \sum_{i=1}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D \Delta y_{\mathcal{B}_i}) + \varepsilon^D l_{\mathcal{N}_s} \\
&= \sum_{i=1}^{n} w_{\mathcal{B}_i} y_{\mathcal{B}_i} + \varepsilon^D \sum_{i=1}^{n} w_{\mathcal{B}_i} \Delta y_{\mathcal{B}_i} + \varepsilon^D l_{\mathcal{N}_s} \\
&= h(y) + \varepsilon^D \left(w_{\mathcal{B}}^T \Delta y_{\mathcal{B}} + l_{\mathcal{N}_s}\right) \\
&= h(y) + \varepsilon^D \left(l_{\mathcal{N}_s} - w_{\mathcal{N}_s}\right), \qquad (3.17)
\end{aligned}
$$

as $w_{\mathcal{B}}^T \Delta y_{\mathcal{B}} = w_{\mathcal{B}}^T (-B^{-1})^T a^{\mathcal{N}_s} = -x^T a^{\mathcal{N}_s} = -w_{\mathcal{N}_s}$. From (3.17), we see that the value of the objective function changes by $\varepsilon^D \theta \geq 0$ after the perturbation, where $\theta := l_{\mathcal{N}_s} - w_{\mathcal{N}_s} > 0$. If $y_{\mathcal{B}} > 0$, then we have $\varepsilon^D > 0$ without causing a change of sign to any of the dual components. In such case, we can strictly improve the value of the objective function after the perturbation. □

From Proposition 3.3.1, we see that if a primal component violates its lower bound, then we can obtain a dual solution with a better objective value, in case the basis is nondegerate. A violated upper bound can also be used with this purpose, as stated in Proposition 3.3.2.

Proposition 3.3.2. *Consider a basis with a feasible dual basic solution y. Suppose that the primal basic solution (x, w) is infeasible with $w_{\mathcal{N}_s} > u_{\mathcal{N}_s}$ for a given $s \in \{1, \ldots, m-n\}$. Let Δy be the* dual search direction *as defined in (3.13). Then, we can obtain a new dual solution $\overline{y} = y - \varepsilon^D \Delta y$, for a given* dual step-size *$\varepsilon^D \geq 0$, so that $\overline{y}$ is feasible and satisfies $h(\overline{y}) \geq h(y)$. In case the current basis is nondegenerate, this inequality holds strictly.*

Proof. This proof follows the same developments as in the proof of Proposition 3.3.1. However, the perturbation of the nonbasic components of the dual solution is given by

$$
\overline{y}_{\mathcal{N}} = y_{\mathcal{N}} - \varepsilon^D e_s = -\varepsilon^D e_s,
$$

As a result, the basic components become

$$
\overline{y}_{\mathcal{B}} = y_{\mathcal{B}} - \varepsilon^D \left(-B^{-1}a^{\mathcal{N}_s}\right), \qquad (3.18)
$$

Hence, the new dual solution is given by $\overline{y} = y - \varepsilon^D \Delta y$. Following the same developments as in (3.16) and (3.17), the change in the objective function value after the perturbation is

$$h(\overline{y}) = h(y) - \varepsilon^D \theta, \tag{3.19}$$

where $\theta := u_{\mathcal{N}_s} - w_{\mathcal{N}_s} < 0$. Therefore, $h(\overline{y}) \geq h(y)$. If the current basis is nondegenerate, then all the dual components are different from zero and hence a positive step-size ε^D can be chosen. In such case, $h(\overline{y}) > h(y)$. □

Propositions 3.3.1 and 3.3.2 show how to obtain a new dual solution by using a step-size ε^D and the dual search direction Δy. However, these propositions do not give an explicit choice for the step-size ε^D. In the proofs, we have seen that the improvement in the objective function value is proportional to ε^D, so we should set ε^D as large as possible. On the other hand, the choice of ε^D is limited by the assumption that the sign of each dual basic component is not changed after the perturbation. As a consequence, to obtain the largest value of ε^D, we have to analyze the sign of each $y_{\mathcal{B}_i}$ and the sign of the corresponding component in the dual search direction Δy. First, consider the case in which the dual solution is perturbed as in Proposition 3.3.1. Suppose that $y_{\mathcal{B}_i} \geq 0$ for a given $i \in \{1, \ldots, n\}$. If $\Delta y_{\mathcal{B}_i} \geq 0$, then $\overline{y}_{\mathcal{B}_i} = y_{\mathcal{B}_i} + \varepsilon^D \Delta y_{\mathcal{B}_i}$ is also nonnegative for any $\varepsilon^D \geq 0$. On the other hand, if $\Delta y_{\mathcal{B}_i} < 0$, then

$$y_{\mathcal{B}_i} + \varepsilon^D \Delta y_{\mathcal{B}_i} \geq 0 \;\Leftrightarrow\; \varepsilon^D \Delta y_{\mathcal{B}_i} \geq -y_{\mathcal{B}_i} \;\Leftrightarrow\; \varepsilon^D \leq -\frac{y_{\mathcal{B}_i}}{\Delta y_{\mathcal{B}_i}}. \tag{3.20}$$

Hence, the largest value we can choose for ε^D is $|y_{\mathcal{B}_i}/\Delta y_{\mathcal{B}_i}|$. Suppose now that $y_{\mathcal{B}_i} \leq 0$. If $\Delta y_{\mathcal{B}_i} \leq 0$, then $\overline{y}_{\mathcal{B}_i}$ is also nonpositive for any $\varepsilon^D \geq 0$. However, for $\Delta y_{\mathcal{B}_i} > 0$ we have

$$y_{\mathcal{B}_i} + \varepsilon^D \Delta y_{\mathcal{B}_i} \leq 0 \;\Leftrightarrow\; \varepsilon^D \Delta y_{\mathcal{B}_i} \leq -y_{\mathcal{B}_i} \;\Leftrightarrow\; \varepsilon^D \leq -\frac{y_{\mathcal{B}_i}}{\Delta y_{\mathcal{B}_i}}, \tag{3.21}$$

and thus ε^D is bounded above by the same amount as in (3.20). Let $\mathcal{P}^+ \subset \mathcal{B}$ be define as

$$\mathcal{P}^+ := \left\{ j \in \mathcal{B} \mid (j \in \mathcal{B}^l \text{ and } \Delta y_j < 0) \text{ or } (j \in \mathcal{B}^u \text{ and } \Delta y_j > 0) \right\}. \tag{3.22}$$

The choice of ε^D must take into account the basic indices that belong to $\mathcal{P}^+$ only. If $\mathcal{P}^+ = \varnothing$, then ε^D can be as large as we may want, without causing any change of sign after the perturbation. In such case, the dual problem is unbounded, so the primal problem can only be infeasible. On the other hand, if $\mathcal{P}^+ \neq \varnothing$, then the largest value that ε^D may assume in order to satisfy (3.20) and (3.21) for all the basic indices in $\mathcal{P}^+$ is given by the *ratio test*:

$$\varepsilon^D \;:=\; \min_{i=1,\ldots,n} \left\{ -\frac{y_{\mathcal{B}_i}}{\Delta y_{\mathcal{B}_i}} \mid \mathcal{B}_i \in \mathcal{P}^+ \right\}. \tag{3.23}$$

Assume that the minimum ratio is obtained for the p-th basic index, *i.e.*, $\varepsilon^D = -y_{\mathcal{B}_p}/\Delta y_{\mathcal{B}_p}$. Hence, we have

$$\overline{y}_{\mathcal{B}_p} = y_{\mathcal{B}_p} + \left(-\frac{y_{\mathcal{B}_p}}{\Delta y_{\mathcal{B}_p}} \right) \Delta y_{\mathcal{B}_p} = 0 \;\text{ and }\; \overline{y}_{\mathcal{N}_s} = -\frac{y_{\mathcal{B}_p}}{\Delta y_{\mathcal{B}_p}} \geq 0.$$

This suggests a *basis change*, as the p-th basic component becomes equal to zero and the s-th nonbasic component becomes positive if the basis is nondegenerate. Thus, we swap indices $\mathcal{B}_p$ and $\mathcal{N}_s$ in order to satisfy Definition 3.2.1. The current index $\mathcal{N}_s$ becomes basic at the

p-th position in $\mathcal{B}$, while the current index $\mathcal{B}_p$ becomes the s-th nonbasic index in $\mathcal{N}$. After the basis change we have $w_{\mathcal{N}_s} = l_{\mathcal{N}_s}$, as this component becomes basic (see Definition 3.2.1). The new sets of basic and nonbasic indices are respectively given by

$$\begin{aligned} \overline{\mathcal{B}} &= \{\mathcal{B}_1, \ldots, \mathcal{B}_{p-1}, \mathcal{N}_s, \mathcal{B}_{p+1}, \ldots, \mathcal{B}_n\}, \\ \overline{\mathcal{N}} &= \{\mathcal{N}_1, \ldots, \mathcal{N}_{s-1}, \mathcal{B}_p, \mathcal{N}_{s+1}, \ldots, \mathcal{N}_{m-n}\}. \end{aligned}$$

A similar result is obtained for the case in which the dual solution is perturbed as in Proposition 3.3.2. Following the same developments as those used to obtain (3.20) and (3.21), we define the subset of basic indices

$$\mathcal{P}^- = \left\{ j \in \mathcal{B} \mid (j \in \mathcal{B}^l \text{ and } \Delta y_j > 0) \text{ or } (j \in \mathcal{B}^u \text{ and } \Delta y_j < 0) \right\}. \tag{3.24}$$

If $\mathcal{P}^- = \varnothing$ then the dual problem is unbounded and therefore the primal problem is infeasible. Otherwise, the largest possible value of ε^D which does not change the sign of $y_{\mathcal{B}_i}$ is given by the ratio test

$$\varepsilon^D = \min_{i=1,\ldots,m} \left\{ \frac{y_{\mathcal{B}_i}}{\Delta y_{\mathcal{B}_i}} \mid \mathcal{B}_i \in \mathcal{P}^- \right\}. \tag{3.25}$$

Assume that the minimum ratio is obtained for the p-th basic index. After applying the perturbation with the ε^D given by the ratio test, we have $\overline{y}_{\mathcal{B}_p} = 0$ and $\overline{y}_{\mathcal{N}_s} \geq 0$. This suggests the basis change in which $\mathcal{N}_s$ enters the basis, while $\mathcal{B}_p$ becomes nonbasic.

The basis change results in a perturbation of the primal basic solution as well. In Proposition 3.3.3, we show how to modify the current primal basic solution in order to reflect this basis change.

Proposition 3.3.3. *Consider a basis which is dual feasible and has an infeasible primal basic solution (x, w) such that $w_{\mathcal{N}_s}$ violates one of its bounds. Assume we perform a basis change in which the current p-th basic index leaves the basis and the current s-th nonbasic index enters the basis. We denote by $(\overline{x}, \overline{w})$ the primal basic solution of the new basis. Let $(\Delta x, \Delta w)$ be the* primal search direction *which is defined as*

$$\Delta x = (B^{-1})^T e_p, \tag{3.26}$$

$$\Delta w = \begin{bmatrix} \Delta w_{\mathcal{B}} \\ \Delta w_{\mathcal{N}} \end{bmatrix} = \begin{bmatrix} e_p \\ N^T \Delta x \end{bmatrix}, \tag{3.27}$$

where e_p is the p-th column of the identity matrix of order n. Then, $(\overline{x}, \overline{w}) := (x, w) + \varepsilon^P(\Delta x, \Delta w)$, where $\varepsilon^P = (\tau - w_{\mathcal{N}_s})/\Delta w_{\mathcal{N}_s}$ is the primal step-size, *with $\tau = l_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$, and $\tau = u_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} > u_{\mathcal{N}_s}$.*

Proof. Since the p-th basic index leaves the basis, $\overline{w}_{\mathcal{B}_p}$ is not required to be equal to one of its bounds. Hence, we modify $w_{\mathcal{B}_p}$ by using a perturbation $\varepsilon^P \neq 0$. The remaining basic components are not modified. Observe that we must allow this perturbation to be negative, as $w_{\mathcal{B}_p}$ may be equal to its upper bound. The new solution is given by $\overline{w}_{\mathcal{B}} = w_{\mathcal{B}} + \varepsilon^P e_p$, where e_p is the p-th column of the identity matrix of order n. This perturbation results in modifying the components of the primal variable x. Indeed, according to the primal general solution which is defined in (3.10), we obtain

$$\begin{aligned} \overline{x} &= (B^{-1})^T \left(w_{\mathcal{B}} + \varepsilon^P e_p \right) \\ &= (B^{-1})^T w_{\mathcal{B}} + (B^{-1})^T \varepsilon^P e_p \\ &= x + \varepsilon^P \Delta x, \end{aligned} \tag{3.28}$$

where Δx is defined in (3.26). Furthermore, still according to the primal general solution, we have that

$$\begin{aligned} \overline{w}_{\mathcal{N}} &= N^T(x + \varepsilon^P \Delta x) \\ &= N^T x + \varepsilon^P N^T \Delta x. \end{aligned}$$

Hence, by using the above results and (3.27) we have that $\overline{w} = w + \varepsilon^P \Delta w$. In addition, since the current s-th nonbasic index enters the basis, the component $\overline{w}_{\mathcal{N}_s}$ must be equal to one of its bounds after the basis changes. Let τ be equal to the bound that is violated by $w_{\mathcal{N}_s}$, *i.e.*, $\tau = l_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$, and $\tau = u_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} > u_{\mathcal{N}_s}$. We must have $\overline{w}_{\mathcal{N}_s} = \tau$ in order to satisfy the Definition 3.2.2. To achieve that, ε^P must satisfy $w_{\mathcal{N}_s} + \varepsilon^P \Delta w_{\mathcal{N}_s} = \tau$, which leads to

$$\varepsilon^P = \frac{(\tau - w_{\mathcal{N}_s})}{\Delta w_{\mathcal{N}_s}}.$$

This ratio is well-defined, as $w_{\mathcal{N}_s} = e_s^T w_{\mathcal{N}} = e_s^T N^T x = (a^{\mathcal{N}_s})^T x = (a^{\mathcal{N}_s})^T (B^{-1})^T e_p = -\Delta y_{\mathcal{B}_p}$. In addition, we have $\varepsilon^P \neq 0$ as $\tau - w_{\mathcal{N}_s} \neq 0$. By using this perturbation value, the s-th component of $w_{\mathcal{N}}$ becomes equal to one of its bound, while the current p-th component of $w_{\mathcal{B}}$ may assume any value. Therefore, we have that $(\overline{x}, \overline{w}) := (x, w) + \varepsilon^P(\Delta x, \Delta w)$ is the primal basic solution of the new basis. □

3.4 Dual simplex method for problems in the general form

In the previous section, we addressed the main properties of basic solutions and showed how a basic solution can be improved by perturbing a nonbasic dual component. These ideas are the key components of a dual simplex-type method. Assume we have an initial basis in which the dual basic solution is feasible, but the primal basic solution violates at least one of its bounds. We can iteratively apply basis changes following the discussion above, until the primal basic solution becomes feasible. When this happens, the method terminates, as the optimal solution has been found (Proposition 3.2.3). In Algorithm 5 we give a formal description of the dual simplex method for problems in the general form.

Algorithm 5 begins with the computation of the primal and the dual basic solutions (lines 2 to 4). Then it performs a series of basis changes iteratively, until one of the following stopping criteria is satisfied: (*i*) the primal basic solution is feasible and hence an optimal solution has been found (line 8); (*ii*) the dual step-size is unbounded, so the problem is infeasible (line 15); (*iii*) the maximum number of iterations is achieved (line 6). The *pricing* operation in line 9 corresponds to select an index to enter the basis, following Propositions 3.3.1 and 3.3.2. After computing the dual search direction in line 10, the algorithm perform the ratio test following (3.23) and (3.25). If $\mathcal{P}^+ \neq \varnothing$ or $\mathcal{P}^- \neq \varnothing$ (depending on how $w_{\mathcal{N}_s}$ violates its bounds), then the dual step-size ε^D is bounded and it suggests a basic index to leave the basis. The basis change is then performed in line 22. In line 21 of Algorithm 5, we update the primal and the dual solutions. Of course, we could recompute them from scratch by using the expressions in lines 2 to 4 of Algorithm 5. However, this would be very expensive, because we have to solve several linear systems. Hence, updating the solutions is important to improve the overall performance of the algorithm. It is worth mentioning that an improved ratio test may be used in Algorithm 5, by exploiting the fact that dual variables y are unbounded (see Section 3.5). In addition, it is important to represent the basic matrix by using the LU factorization in order to solve the linear systems efficiently. This issue is discussed in Section 3.6.2.

We close this section by mentioning the main differences between the dual simplex method for problems in the general form and the standard dual simplex method (designed for problems

Algorithm 5: Dual simplex method for problems in the general form.

Input: matrix A; vectors c, l and u; basic partition $A^T = [B \mid N]$ which is dual feasible; IT_MAX.
Output: optimal solution x; or it detects the problem is infeasible; or IT reaches IT_MAX.

IT = 0;
Compute the dual basic solution: $y_{\mathcal{B}} := B^{-1}c$ and $y_{\mathcal{N}} := 0$;
For each $i \in \{1, \ldots, n\}$ set $w_{\mathcal{B}_i} := l_{\mathcal{B}_i}$ if $y_{\mathcal{B}_i} \geq 0$, and $w_{\mathcal{B}_i} := u_{\mathcal{B}_i}$ otherwise;
Compute the primal basic solution: $x := (B^{-1})^T w_{\mathcal{B}}$ and $w_{\mathcal{N}} := N^T x$;

While (IT < IT_MAX) do
{
    If ($l_{\mathcal{N}} \leq w_{\mathcal{N}} \leq u_{\mathcal{N}}$) then STOP, *an optimal solution has been found*;
    Pricing operation: choose an index s such that $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$ or $w_{\mathcal{N}_s} > u_{\mathcal{N}_s}$;
    Compute the dual search direction: $\Delta y_{\mathcal{B}} := -B^{-1}a^{\mathcal{N}_s}$ and $\Delta y_{\mathcal{N}} := e_s$;

    Set $p := -1$;
    If ($w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$) then $p := \arg\min_{i=1,\ldots,n} \left\{-y_{\mathcal{B}_i}/\Delta y_{\mathcal{B}_i} \mid \mathcal{B}_i \in \mathcal{P}^+\right\}$;
    Else $p := \arg\min_{i=1,\ldots,n} \left\{y_{\mathcal{B}_i}/\Delta y_{\mathcal{B}_i} \mid \mathcal{B}_i \in \mathcal{P}^-\right\}$;
    If ($p = -1$) then STOP, *the problem is infeasible*;

    Compute the primal search direction: $\Delta x := (B^{-1})^T e_p$, $\Delta w_{\mathcal{B}} := e_p$ and $\Delta w_{\mathcal{N}} := N^T \Delta x$;
    If ($w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$) then $\tau = l_{\mathcal{N}_s}$ else $\tau = u_{\mathcal{N}_s}$;
    Compute the primal step-size: $\varepsilon^P = (\tau - w_{\mathcal{N}_s})/\Delta w_{\mathcal{N}_s}$;

    Update the current solution: $(x, w, y) := (x, w, y) + (\varepsilon^P \Delta x, \varepsilon^P \Delta w, \varepsilon^D \Delta y)$;
    Basis change: $\mathcal{N}_s$ becomes the p-th basic index and $\mathcal{B}_p$ becomes the s-th nonbasic index;
    IT = IT + 1;
}

in the standard form) as presented by Maros (2003a) and Koberstein (2008). In the variant for problems in the general form, we have a basic matrix of order n, while in the standard dual simplex method the basic matrix is of order $\overline{m}$ (where $\overline{m}$ is the number of nontrivial constraints, *i.e.*, the constraints that cannot be written as variable bounds $l_j \leq x_j \leq u_j$). This would be a serious drawback of this variant in relation to the standard approach, specially in formulations with $n \gg \overline{m}$. However, we consider a computational implementation of the method which relies in the LU factorization of the basic matrix, following the clever techniques proposed by Suhl and Suhl (1990). In such case, the trivial rows/columns of the basic matrix are handled implicitly by the factorization. In other words, the factorization is applied only to the submatrix corresponding to nontrivial constraints (*i.e.*, the same $\overline{m}$ constraints as in the standard method). The LU factorization works similarly in the standard dual simplex method. As a result, we have similar computational performances in practice for both methods regarding the factorization of the basic matrix and its use to solve linear systems.

Another interesting feature of the dual simplex method for problems in the general form is to allow the insertion of a new constraint $l_j \leq a^j x \leq u_j$ without changing the dimension of the basic matrix. This is important when the simplex method is used to solve a sequence of closely-related linear programming problems, a common scenario when using an integer programming methodology. For instance, a new constraint of this type may be generated after branching in the branch-and-bound method, or after the separation subproblem is called in the cutting plane method. Furthermore, many constraints may be generated at a time and added to the problem without requiring a new factorization of the basis matrix. In the standard dual simplex method, the basic matrix must be modified every time a new constraint is added to the problem.

3.5 The bound flipping ratio test

In Section 3.3, we presented the ratio test as a way to obtain the maximum value for the dual step-size ε^D. The idea behind the test was that no dual component could change its sign after the basis change. However, notice that the dual component y_j is free, for each $j \in \mathcal{J}^b$. In such case, we can allow these components to change their signs, without violating any definition. Then, we may obtain larger step-sizes, which are likely to result in larger improvements of the value of the dual objective function. This idea results in the *bound-flip ratio test*, which we describe in this section.

Consider a basis change according to Proposition 3.3.1, in which the primal component $w_{\mathcal{N}_s}$ violates its lower bound $l_{\mathcal{N}_s}$, for a given index $s \in \{1, \ldots, m-n\}$. Let ε_1^D be the largest step-size so that the sign of each dual component remains the same after the basis change. Following the developments in Section 3.3, ε_1^D can be determined by the ratio test

$$\varepsilon_1^D \;=\; \min_{i=1,\ldots,n} \left\{ -\frac{y_{\mathcal{B}_i}}{\Delta y_{\mathcal{B}_i}} \mid \mathcal{B}_i \in \mathcal{P}^+ \right\}, \tag{3.29}$$

where $\mathcal{P}^+$ is defined as in (3.22). If $\mathcal{P}^+ \neq \varnothing$, then we can compute ε_1^D according to a given index $\mathcal{B}_p \in \mathcal{P}^+$ that satisfies the minimization in the ratio test. By using ε_1^D as the dual step-size, we obtain the new dual solution $y^1 = y + \varepsilon_1^D \Delta y$, with the objective function value

$$h(y^1) = \sum_{i=1}^{n} h_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_1^D \Delta y_{\mathcal{B}_i}) + \varepsilon_1^D l_{\mathcal{N}_s} = h(y) + \varepsilon_1^D \left(l_{\mathcal{N}_s} - w_{\mathcal{N}_s} \right), \tag{3.30}$$

as obtained in (3.17). Recall that in this solution we have $\overline{y}_{\mathcal{B}_p} = 0$, as any step-size $\varepsilon^D > \varepsilon_1^D$ would result in $\text{sign}(\overline{y}_{\mathcal{B}_p}) \neq \text{sign}(y_{\mathcal{B}_p})$. In a degenerate basis, other dual components would also change their signs with a step-size $\varepsilon^D > \varepsilon_1^D$. To make the following discussion easier to understand, we assume for a while that the bases are nondegenerate.

In case the index $\mathcal{B}_p$ belongs to $\mathcal{J}^b$, then a step-size $\varepsilon^D > \varepsilon_1^D$ may be chosen. Let $\varepsilon_2^D = \varepsilon_1^D + \delta$ be a step-size larger than ε_1^D, where $\delta > 0$ is the largest amount so that only the component $\overline{y}_{\mathcal{B}_p}$ has a change of sign. We perturb the dual solution y using this step-size instead of ε_1^D. The resulting dual solution is

$$y^2 \;=\; y + \varepsilon_2^D \Delta y \;=\; y + \varepsilon_1^D \Delta y + \delta \Delta y,$$

such that $\text{sign}(y^2_{\mathcal{B}_p}) \neq \text{sign}(y_{\mathcal{B}_p})$. Due to this change of sign, the variable $w_{\mathcal{B}_p}$ must be reset to its opposite bound to be in accordance to Definition 3.2.2. Hence, we say that a *bound flip* is applied to variable $w_{\mathcal{B}_p}$. Formally, we define the function

$$\text{flip}(w_j) = \begin{cases} l_j, & \text{if } w_j = u_j, \\ u_j, & \text{if } w_j = l_j, \end{cases} \qquad j \in \mathcal{J},$$

which returns the opposite bound in relation to the bound that w_j is currently equal to. With this definition, the value of the objective function given by the new dual solution y^2 is

$$\begin{aligned} h(y^2) \;&=\; \sum_{i=1}^{n} h_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_2^D \Delta y_{\mathcal{B}_i}) + \varepsilon_2^D l_{\mathcal{N}_s} \\ &=\; \sum_{\substack{i=1 \\ i \neq p}}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_2^D \Delta y_{\mathcal{B}_i}) + \text{flip}(w_{\mathcal{B}_p})(y_{\mathcal{B}_p} + \varepsilon_2^D \Delta y_{\mathcal{B}_p}) + \varepsilon_1^D l_{\mathcal{N}_s} + \delta l_{\mathcal{N}_s}. \end{aligned} \tag{3.31}$$

For the step-size ε_1^D we have $y_{\mathcal{B}_p} + \varepsilon_1^D \Delta y_{\mathcal{B}_p} = 0$. Hence, the first term in expression (3.31) can be rewritten as

$$\begin{aligned}
\sum_{\substack{i=1\\ i\neq p}}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_2^D \Delta y_{\mathcal{B}_i}) &= \sum_{i=1}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_1^D \Delta y_{\mathcal{B}_i}) + \delta \sum_{\substack{i=1\\ i\neq p}}^{n} w_{\mathcal{B}_i} \Delta y_{\mathcal{B}_i} \\
&= \sum_{i=1}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_1^D \Delta y_{\mathcal{B}_i}) + \delta \sum_{i=1}^{n} w_{\mathcal{B}_i} \Delta y_{\mathcal{B}_i} - \delta w_{\mathcal{B}_p} \Delta y_{\mathcal{B}_p} \\
&= \sum_{i=1}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_1^D \Delta y_{\mathcal{B}_i}) + \delta(w_{\mathcal{B}}^T \Delta y_{\mathcal{B}} - w_{\mathcal{B}_p} \Delta y_{\mathcal{B}_p}) \\
&= \sum_{i=1}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_1^D \Delta y_{\mathcal{B}_i}) + \delta(-w_q - w_{\mathcal{B}_p} \Delta y_{\mathcal{B}_p}),
\end{aligned}$$

where $w_{\mathcal{B}}^T \Delta y_{\mathcal{B}} = w_{\mathcal{B}}^T (-B^{-1})^T a^{\mathcal{N}_s} = -x^T a^{\mathcal{N}_s} = -w_{\mathcal{N}_s}$ was used to obtain the last equality. For the second term in expression (3.31) we have

$$\text{flip}(w_{\mathcal{B}_p})(y_{\mathcal{B}_p} + \varepsilon_2^D \Delta y_{\mathcal{B}_p}) = \text{flip}(w_{\mathcal{B}_p})(y_{\mathcal{B}_p} + \varepsilon_1^D \Delta y_{\mathcal{B}_p} + \delta \Delta y_{\mathcal{B}_i}) = \delta \text{ flip}(w_{\mathcal{B}_p}) \Delta y_{\mathcal{B}_p}.$$

If we replace the two first terms in (3.31) by using the obtained expressions, the new value of the dual objective function is

$$\begin{aligned}
h(y^2) &= \sum_{i=1}^{n} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_1^D \Delta y_{\mathcal{B}_i}) + \varepsilon_1^D l_{\mathcal{N}_s} + \delta(-w_{\mathcal{N}_s} - w_{\mathcal{B}_p} \Delta y_{\mathcal{B}_p} + \text{flip}(w_{\mathcal{B}_p}) \Delta y_{\mathcal{B}_p} + l_{\mathcal{N}_s}) \\
&= h(y^1) + \delta(-w_{\mathcal{N}_s} - w_{\mathcal{B}_p} \Delta y_{\mathcal{B}_p} + \text{flip}(w_{\mathcal{B}_p}) \Delta y_{\mathcal{B}_p} + l_{\mathcal{N}_s}) \\
&= h(y^1) + \delta(l_{\mathcal{N}_s} - w_{\mathcal{N}_s}) + \delta(\text{flip}(w_{\mathcal{B}_p}) - w_{\mathcal{B}_p}) \Delta y_{\mathcal{B}_p} \\
&= h(y^1) + \delta\theta^1 + \delta(\text{flip}(w_{\mathcal{B}_p}) - w_{\mathcal{B}_p}) \Delta y_{\mathcal{B}_p},
\end{aligned}$$

where $\theta^1 := l_{\mathcal{N}_s} - w_{\mathcal{N}_s}$. We can further simplify the last expression, by noticing that

$$(\text{flip}(w_{\mathcal{B}_p}) - w_{\mathcal{B}_p}) \Delta y_{\mathcal{B}_p} = -(u_{\mathcal{B}_p} - l_{\mathcal{B}_p}) |\Delta y_{\mathcal{B}_p}|.$$

Indeed, from (3.22) and (3.23), we have that $y_{\mathcal{B}_p} \geq 0$ implies in $\Delta y_{\mathcal{B}_p} < 0$ and $w_{\mathcal{B}_p} = l_{\mathcal{B}_p}$. On the other hand, $y_{\mathcal{B}_p} \leq 0$ implies in $\Delta y_{\mathcal{B}_p} > 0$ and $w_{\mathcal{B}_p} = u_{\mathcal{B}_p}$. Hence, the above equality holds in both cases. Finally, we obtain

$$\begin{aligned}
h(y^2) &= h(y^1) + \delta\theta^1 - \delta(u_{\mathcal{B}_p} - l_{\mathcal{B}_p}) |\Delta y_{\mathcal{B}_p}| \\
&= h(y^1) + \delta\theta^2, &(3.32)\\
&= h(y^1) + (\varepsilon_2^D - \varepsilon_1^D)\theta^2, &(3.33)
\end{aligned}$$

where $\theta^2 := \theta^1 - (u_{\mathcal{B}_p} - l_{\mathcal{B}_p}) |\Delta y_{\mathcal{B}_p}|$. In summary, by using the step-size $\varepsilon_2^D > \varepsilon_1^D$ we further modify the value of objective function by the amount $(\varepsilon_2^D - \varepsilon_1^D)\theta^2$. Therefore, if $\theta^2 \geq 0$, then a larger improvement is achieved so that $h(y^2) \geq h(y^1) \geq h(y)$.

Recall that we chose ε_2^D as the largest step-size such that only $y_{\mathcal{B}_p}$ would change its sign. To obtain this value, we can also use the ratio test given by (3.20) and (3.21), for each $i = 1, \ldots, p-1, p+1, \ldots, n$, *i.e.*, without considering $i = p$. This way, the largest value of ε_2^D is given by

$$\varepsilon_2^D = -\frac{y_{\mathcal{B}_r}}{\Delta y_{\mathcal{B}_r}} := \min_{\substack{i=1,\ldots,n\\ i\neq p}} \left\{ -\frac{y_{\mathcal{B}_i}}{\Delta y_{\mathcal{B}_i}} \mid \mathcal{B}_i \in \mathcal{P}^+ \right\}. \tag{3.34}$$

If $\mathcal{P}^+ \setminus \{\mathcal{B}_p\} = \varnothing$, then ε_2^D is unbounded and hence primal problem is infeasible. It is worth mentioning that all the ratios in (3.34) need to be computed to obtain $\mathcal{B}_p$ in the previous ratio test given by (3.29). ε_2^D corresponds to the second smallest ratio in (3.29).

Fig. 3.2 illustrates the behavior of the dual objective function in relation to the step-size ε^D (assuming nondegenerate bases). In part (a), the slope of $h(y)$ is equal to θ^1 for $\varepsilon^D < \varepsilon_1^D$, as obtained in (3.30). For $\varepsilon_1^D < \varepsilon^D < \varepsilon_2^D$, the slope is reduced to $\theta^2 = \theta^1 - (u_{\mathcal{B}_p} - l_{\mathcal{B}_p})|\Delta y_{\mathcal{B}_p}|$, as presented in (3.32). ε_1^D and ε_2^D are breakpoints of $h(y)$, in which the slope of the function changes. The reason for these changes is illustrated in parts (b) and (c). In part (b), we see that for $y_{\mathcal{B}_p} < 0$, the contribution of $h_{\mathcal{B}_p}(y_{\mathcal{B}_p})$ to the dual objective function is $u_{\mathcal{B}_p} y_{\mathcal{B}_p}$. For $\varepsilon^D = \varepsilon_1^D$, there is a breakpoint of $h_{\mathcal{B}_p}(y_{\mathcal{B}_p})$, because $y_{\mathcal{B}_p} + \varepsilon_1^D \Delta y_{\mathcal{B}_p} = 0$. After this point, the slope of $h_{\mathcal{B}_p}(y_{\mathcal{B}_p})$ becomes $l_{\mathcal{B}_p}$. Hence, the difference in relation to the previous slope is $(u_{\mathcal{B}_p} - l_{\mathcal{B}_p})$. Since no other variable changes its sign for $\varepsilon^D \leq \varepsilon_2^D$, the slope of $h(y)$ changes by $(u_{\mathcal{B}_p} - l_{\mathcal{B}_p})|\Delta y_{\mathcal{B}_p}|$. The behavior of $y_{\mathcal{B}_r}$ is similar, but the breakpoint of $h_{\mathcal{B}_r}(y_{\mathcal{B}_r})$ is given by $\varepsilon^D = \varepsilon_2^D$, as illustrated in part (c).

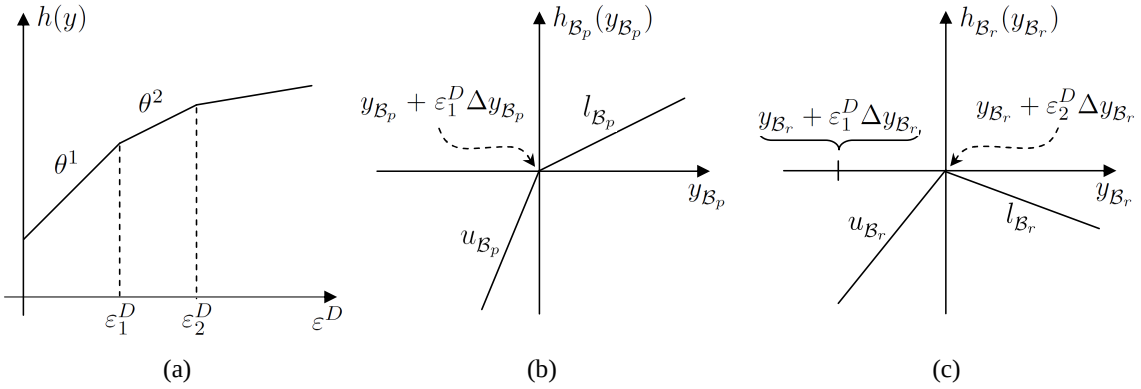

Figure 3.2: Illustration of the bevahior of the dual objective function according to the variation of the step-size ε^D. (a) Dual objective function $h(y)$ with breakpoints at $\varepsilon^D = \varepsilon_1^D$ and $\varepsilon^D = \varepsilon_2^D$. (b) Function $h_{\mathcal{B}_p}(y_{\mathcal{B}_p})$ with $y_{\mathcal{B}_p}$ as a function of ε^D. (c) Function $h_{\mathcal{B}_r}(y_{\mathcal{B}_r})$ with $y_{\mathcal{B}_r}$ as a function of ε^D.

If $\mathcal{B}_r \in \mathcal{J}^b$, then we can apply the same ideas used so far to verify whether a step-size larger than ε_2^D would further improve the value of the dual objective function. Actually, these ideas can be applied repeatedly as long as the variables are bounded and the slope of $h(y)$ is positive. As we increase the step-size ε^D, we change the signs of more and more components $y_{\mathcal{B}_i}$ with $\mathcal{B}_i \in \mathcal{P}^+$. The values of ε^D which cause the change of signs are the breakpoints of the dual objective function, which are given by

$$\{\varepsilon_1^D, \varepsilon_2^D, \ldots, \varepsilon_t^D\} \;=\; \left\{ -\frac{y_{\mathcal{P}_1^+}}{\Delta y_{\mathcal{P}_1^+}}, -\frac{y_{\mathcal{P}_2^+}}{\Delta y_{\mathcal{P}_2^+}}, \ldots, -\frac{y_{\mathcal{P}_t^+}}{\Delta y_{\mathcal{P}_t^+}} \right\}, \quad t = |\mathcal{P}^+|, \tag{3.35}$$

where we have assumed that the indices in $\mathcal{P}^+$ are sorted so that $\varepsilon_1^D \leq \varepsilon_2^D \leq \ldots \leq \varepsilon_t^D$, without loss of generality. This discussion is formalized in Theorem 3.5.1.

Theorem 3.5.1 (Bound flipping ratio test)**.** *Given a basic partition $A^T = [B \mid N]$ of the coefficient matrix of problem (3.8), consider a basis change which is motivated by the infeasible primal component $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$. Given the current dual basic solution y and the dual search direction Δy which is defined in (3.13), let $\overline{y} = y + \varepsilon^D \Delta y$ denote the dual basic solution after*

the basis change. Consider the ordered set of breakpoints of the dual objective function which is defined in (3.35) with $t = |\mathcal{P}^+| > 1$. *Then, the largest step-size* ε^D *such that the value of the dual objective function increases the most and* $\overline{y}$ *remains feasible, is given by* $\varepsilon^D = \varepsilon^D_k$, $1 \leq k \leq t$, *where* k *is the smallest index which satisfies*

$$(\mathcal{P}^+_k \notin \mathcal{I}^b) \quad or \quad (\theta^k \geq 0 \quad and \quad \theta^{k+1} < 0),$$

where $\theta^{i+1} = \theta^i - (u_{\mathcal{P}^+_i} - l_{\mathcal{P}^+_i})|\eta_{\mathcal{P}^+_i}|$, $\mathcal{P}^+_i \in \mathcal{I}^b$, $i = 1, \ldots, k$, *and* $\theta^1 = l_{\mathcal{N}_s} - w_{\mathcal{N}_s}$.

Proof. From (3.30), we have $h(y^1) = h(y) + \varepsilon^D_1\theta^1$. In addition, we have obtained in (3.33) that for $k = 1$, $h(y^2) = h(y^1) + (\varepsilon^D_2 - \varepsilon^D_1)\theta^2$, with $\theta^2 = \theta^1 - (u_{\mathcal{B}_p} - l_{\mathcal{B}_p})|\Delta y_{\mathcal{B}_p}|$ and $\mathcal{P}^+_1 = \mathcal{B}_p$. Consider now that the result is valid for a given k such that $1 \leq k < t$. Let us define two subsets of breakpoints given by $\mathcal{F} = \{\mathcal{P}^+_1, \ldots, \mathcal{P}^+_{k-1}, \mathcal{P}^+_k\}$ and $\mathcal{F}' = \{\mathcal{P}^+_1, \ldots, \mathcal{P}^+_{k-1}\}$. We assume that $y^k = y + \varepsilon^D_k \Delta y$ is feasible and $\mathcal{F}' \subset \mathcal{I}^b$ (otherwise a smaller k would be taken in order to satisfy this assumption). The k-th breakpoint leads to $y_{\mathcal{F}_k} + \varepsilon^D_k \Delta y_{\mathcal{F}_k} = 0$. If $\mathcal{F}_k \notin \mathcal{I}^b$, then a step-size larger than ε^D_k would make $\overline{y}_{\mathcal{F}_k}$ infeasible and, hence, $\varepsilon^D = \varepsilon^D_k$ is the largest possible step-size. Otherwise, the next breakpoint ε^D_{k+1} would be adopted as the step-size, in case it does not deteriorate the value of the objective function. To verify that, we define the scalar $\varepsilon \geq 0$, so that $\varepsilon^D_{k+1} = \varepsilon^D_k + \varepsilon$. The choice of ε must guarantee that only the dual variables $y_{\mathcal{F}_1}, \ldots, y_{\mathcal{F}_k}$ change their sign. As a consequence, only the primal components $w_{\mathcal{F}_1}, \ldots, w_{\mathcal{F}_k}$ can flip bounds. The value of the dual objective function corresponding to $y^{k+1} = y + \varepsilon^D_{k+1}\Delta y$ is given by

$$\begin{aligned} h(y^{k+1}) &= \sum_{i=1}^{n} h_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_{k+1}\Delta y_{\mathcal{B}_i}) + \varepsilon^D_{k+1} l_{\mathcal{N}_s} \\ &= \sum_{\mathcal{B}_i \notin \mathcal{F}} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_{k+1}\Delta y_{\mathcal{B}_i}) + \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i})(y_{\mathcal{B}_i} + \varepsilon^D_{k+1}\Delta y_{\mathcal{B}_i}) + (\varepsilon^D_k + \varepsilon) l_{\mathcal{N}_s} \end{aligned} \tag{3.36}$$

Notice the similarity between expressions (3.36) and (3.31). Since $y_{\mathcal{F}_k} + \varepsilon^D_k \Delta y_{\mathcal{F}_k} = 0$, we can rewrite the first term in (3.36) as

$$\begin{aligned} \sum_{\mathcal{B}_i \notin \mathcal{F}} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_{k+1}\Delta y_{\mathcal{B}_i}) &= \sum_{\mathcal{B}_i \notin \mathcal{F}'} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_k \Delta y_{\mathcal{B}_i}) + \varepsilon \sum_{\mathcal{B}_i \notin \mathcal{F}} w_{\mathcal{B}_i}\Delta y_{\mathcal{B}_i} \\ &= \sum_{\mathcal{B}_i \notin \mathcal{F}'} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_k \Delta y_{\mathcal{B}_i}) + \varepsilon \sum_{i=1}^{n} w_{\mathcal{B}_i}\Delta y_{\mathcal{B}_i} - \varepsilon \sum_{\mathcal{B}_i \in \mathcal{F}} w_{\mathcal{B}_i}\Delta y_{\mathcal{B}_i} \\ &= \sum_{\mathcal{B}_i \notin \mathcal{F}'} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_k \Delta y_{\mathcal{B}_i}) + \varepsilon \left(w^T_{\mathcal{B}}\Delta\overline{y}_{\mathcal{B}} - \sum_{\mathcal{B}_i \in \mathcal{F}} w_{\mathcal{B}_i}\Delta y_{\mathcal{B}_i} \right) \\ &= \sum_{\mathcal{B}_i \notin \mathcal{F}'} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon^D_k \Delta y_{\mathcal{B}_i}) + \varepsilon \left(-w_{\mathcal{N}_s} - \sum_{\mathcal{B}_i \in \mathcal{F}} w_{\mathcal{B}_i}\Delta y_{\mathcal{B}_i} \right). \end{aligned}$$

We rewrite the second term in (3.36) as

$$\begin{aligned} \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i})(y_{\mathcal{B}_i} + \varepsilon^D_{k+1}\Delta y_{\mathcal{B}_i}) &= \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i})(y_{\mathcal{B}_i} + \varepsilon^D_k \Delta y_{\mathcal{B}_i}) + \varepsilon \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i})\Delta y_{\mathcal{B}_i}. \\ &= \sum_{\mathcal{B}_i \in \mathcal{F}'} \text{flip}(w_{\mathcal{B}_i})(y_{\mathcal{B}_i} + \varepsilon^D_k \Delta y_{\mathcal{B}_i}) + \varepsilon \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i})\Delta y_{\mathcal{B}_i}. \end{aligned}$$

Replacing these two last results in (3.36), we obtain

$$\begin{aligned}
h(y^{k+1}) &= \sum_{\mathcal{B}_i \notin \mathcal{F}'} w_{\mathcal{B}_i}(y_{\mathcal{B}_i} + \varepsilon_k^D \Delta y_{\mathcal{B}_i}) + \sum_{\mathcal{B}_i \in \mathcal{F}'} \text{flip}(w_{\mathcal{B}_i})(y_{\mathcal{B}_i} + \varepsilon_k^D \Delta y_{\mathcal{B}_i}) \\
&\quad + \varepsilon \left(-w_{\mathcal{N}_s} - \sum_{\mathcal{B}_i \in \mathcal{F}} w_{\mathcal{B}_i} \Delta y_{\mathcal{B}_i} + \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i}) \Delta y_{\mathcal{B}_i} + l_{\mathcal{N}_s} \right) + \varepsilon_k^D l_{\mathcal{N}_s} \\
&= h(y^k) + \varepsilon \left(-w_{\mathcal{N}_s} - \sum_{\mathcal{B}_i \in \mathcal{F}} w_{\mathcal{B}_i} \Delta y_{\mathcal{B}_i} + \sum_{\mathcal{B}_i \in \mathcal{F}} \text{flip}(w_{\mathcal{B}_i}) \Delta y_{\mathcal{B}_i} + l_{\mathcal{N}_s} \right) \\
&= h(y^k) + \varepsilon(l_{\mathcal{N}_s} - w_{\mathcal{N}_s}) + \varepsilon \sum_{\mathcal{B}_i \in \mathcal{F}} (\text{flip}(w_{\mathcal{B}_i}) - w_{\mathcal{B}_i}) \Delta y_{\mathcal{B}_i} \\
&= h(y^k) + \varepsilon(l_{\mathcal{N}_s} - w_{\mathcal{N}_s}) - \varepsilon \sum_{\mathcal{B}_i \in \mathcal{F}'} (u_{\mathcal{B}_i} - l_{\mathcal{B}_i}) |\Delta y_{\mathcal{B}_i}| - \varepsilon(u_{\mathcal{F}_k} - l_{\mathcal{F}_k}) |\Delta y_{\mathcal{F}_k}| \\
&= h(y^k) + \varepsilon\theta^k - \varepsilon(u_{\mathcal{F}_k} - l_{\mathcal{F}_k}) |\Delta y_{\mathcal{F}_k}|, \\
&= h(y^k) + \varepsilon\theta^{k+1}, \\
&= h(y^k) + (\varepsilon_{k+1}^D - \varepsilon_k^D)\theta^{k+1}, \qquad (3.37)
\end{aligned}$$

where $\theta^{k+1} := \theta^k - (u_{\mathcal{F}_k} - l_{\mathcal{F}_k})|\Delta y_{\mathcal{F}_k}|$. Therefore, using the step-size ε_{k+1}^D we add the amount $(\varepsilon_{k+1}^D - \varepsilon_k^D)\theta^{k+1}$ to $h(y^k)$. If $\theta^{k+1} \geq 0$, then by choosing $\varepsilon^D = \varepsilon_{k+1}^D$ the value of the objective function improves (or remains the same). Otherwise, this step-size deteriorates the value of the objective, so the best step-size is $\varepsilon^D = \varepsilon_k^D$. □

Fig. 3.3 illustrates Theorem 3.5.1 for $t = 4$ and $k = 3$. The initial slope of $h(y)$ is equal to θ^1. At ε_1^D we see the first breakpoint of $h(y)$, in which $y_{\mathcal{P}_1^+}$ changes its sign. Then, the slope of the dual function is reduced by the amount $(u_{\mathcal{P}_i^+} - l_{\mathcal{P}_i^+})|\Delta y_{\mathcal{P}_i^+}|$, so that it is equal to θ^2 until the next breakpoint ε_2^D. After $\varepsilon^D = \varepsilon_3^D$, the sign of $y_{\mathcal{P}_3^+}$ changes and $h(y)$ starts decreasing as ε^D increases. Between ε_3^D and ε_4^D, the slope of $h(y)$ is $\theta^4 < 0$ and therefore ε_3^D is the dual step-size that results in the largest increase of $h(y)$.

Recall that in the beginning of this section, we assumed that the basis change follows Proposition 3.3.1. Nevertheless, the extension of this analysis is straightforward to the case addressed in Proposition 3.3.2. The dual solution after the basis change is given by $y = y - \varepsilon^D \Delta y$. Analogously to (3.35), the set of breakpoints is given by

$$\{\varepsilon_1^D, \varepsilon_2^D, \ldots, \varepsilon_t^D\} = \left\{ \frac{y_{\mathcal{P}_1^-}}{\Delta y_{\mathcal{P}_1^-}}, \frac{y_{\mathcal{P}_2^-}}{\Delta y_{\mathcal{P}_2^-}}, \ldots, \frac{y_{\mathcal{P}_t^-}}{\Delta y_{\mathcal{P}_t^-}} \right\}, \quad t = |\mathcal{P}^-|, \qquad (3.38)$$

where the indices $\mathcal{P}^-$ are obtained by (3.24). In addition, we assume they are sorted such that $\varepsilon_1^D \leq \varepsilon_2^D \leq \ldots \leq \varepsilon_t^D$.

In summary, Theorem 3.5.1 gives an improved strategy for the ratio test operation by exploiting additional breakpoints of the objective function. In contrast, the standard ratio test uses only the first breakpoint, which results in a smaller (or equal) improvement of the objective function value. The resulting strategy is often called as the *bound flipping ratio test*, due to behavior of the bounds of the primal variables $w_{\mathcal{B}_i}$ when the corresponding components of the dual solution change their sign. As observed in other variants of the dual simplex method, the bound flipping ratio test is a technique that strongly contributes with the performance of the method (see e.g. Maros, 2003a; Koberstein, 2008). This strategy is

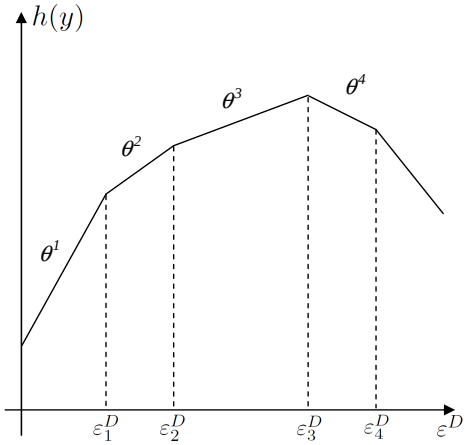

Figure 3.3: Illustration of Theorem 3.5.1 with $t = 4$, $k = 3$ and $\mathcal{P}_i^+ \in \mathcal{I}^b$, for $i = 1, \ldots, 4$. The breakpoints $\varepsilon_1^D, \ldots, \varepsilon_4^D$ are determined by the changes of the sign of components $y_{\mathcal{P}_1^+}, \ldots, y_{\mathcal{P}_4^+}$. After $\varepsilon^D = \varepsilon_3^D$, $h(y)$ decreases as ε^D increase, which indicates that ε_3^D is the step-size that leads to the largest improvement in the value of the objective function.

recognized in the literature as one of the reasons of having the dual variants as better choices in relation to the primal simplex-type methods in practice. Indeed, no equivalent feature is available in the primal variants.

Apart from typically reducing the number of iterations, the bound flipping ratio test also contributes with the numerical stability of the dual simplex method. In the discussion presented above, the step-size $\varepsilon^D = \varepsilon_k^D$ is chosen because it leads to the largest improvement of $h(y)$. However, it may cause numerical instability if the resulting pivot $\Delta y_{\mathcal{P}_k^+}$ is too small. In such case, the step-size should be chosen by recurring to any other breakpoint smaller than ε_k^D, with a more appropriate pivot value.

Finally, it is worth mentioning that the bound flipping ratio test also helps to get rid of dual degeneracy, as we typically obtain larger step-sizes. Indeed, all the analysis that we have presented so far can be extended to the case with degenerate dual bases. The difference is that we may have breakpoints that lead to zero step-sizes. For example, suppose we have a degenerate dual basic solution which has only one basic component that is equal to zero (thus, it is degenerate). In addition, assume the index of this component is active in the ratio test, *i.e.*, it belongs either to $\mathcal{P}^+$ or to $\mathcal{P}^-$. As a consequence, this component will result in the minimum ratio at the ratio test, which yield a step-size equal to zero. Therefore, the objective function does not improve if we select this component to leave the basis. In general, if $l > 0$ basic components of the dual solution which are equal to zero are active in the ratio test (*i.e.*, they belong to either $\mathcal{P}^+$ or $\mathcal{P}^-$), then the first l breakpoints result in step-sizes that are equal to zero. If $l < k$, then the breakpoint chosen in the ratio test is not related to degenerate components and hence it leads to a positive step-size. In such case, the bound flipping ratio test was useful to overcome the degeneracy of the dual solution.

We close this section by proposing how to update the primal solution after a basis change. When using the bound flipping ratio test, the dual solution can be updated as discussed in Section 3.3. However, to update the primal solution we must take into account the bound flip of variables $w_{\mathcal{B}_i}$, for each $\mathcal{B}_i \in \mathcal{F}$. In the following discussion, we adopt the same notation as in Section 3.3. In addition, we use the set $\mathcal{F}$ as defined in the proof of Theorem 3.5.1. This

set consists of the basic indices associated to the smallest breakpoints of the dual objective function.

Proposition 3.5.2. *Consider a basis which is dual feasible and has an infeasible primal basic solution (x, w) in which $w_{\mathcal{N}_s}$ violates one of its bounds. Let τ be defined as $\tau = l_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$, and $\tau = u_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} > u_{\mathcal{N}_s}$. Assume we perform a basis change in which the current p-th basic index leaves the basis and the current s-th nonbasic index enters the basis. Assume also that the index p is chosen by using the bound flipping ratio test, such that the set $\mathcal{F}$ contains the k basic indices associated to a bound flip. Let $(\overline{x}, \overline{w})$ denote the primal basic solution of the new basis. Then, $(\overline{x}, \overline{w}) := (x, w) + \varepsilon^P(\Delta x, \Delta w)$, where $\varepsilon^P = (\tau - w_{\mathcal{N}_s})/\Delta w_{\mathcal{N}_s}$ is the* primal step-size, *and $(\Delta x, \Delta w)$ is the primal search direction which is given by*

$$\Delta w_{\mathcal{B}_i} = \begin{cases} 1, & \text{if } i = p, \\ (u_{\mathcal{B}_i} - l_{\mathcal{B}_i})/\epsilon^P, & \text{if } \mathcal{B}_i \in \mathcal{F} \cap \mathcal{B}^l, \\ (l_{\mathcal{B}_i} - u_{\mathcal{B}_i})/\epsilon^P, & \text{if } \mathcal{B}_i \in \mathcal{F} \cap \mathcal{B}^u, \\ 0, & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, n, \tag{3.39}$$

$\Delta x = (B^{-1})^T \Delta w_{\mathcal{B}}$ *and* $\Delta w_{\mathcal{N}} = N^T \Delta x$.

Proof. The current p-th basic index is chosen to leave the basis, so $\overline{w}_{\mathcal{B}_p}$ does not have to be equal to one of its bounds after the basis change. Hence, we perturb $w_{\mathcal{B}_p}$ by using a value $\varepsilon^P \neq 0$, so that $\overline{w}_{\mathcal{B}_p} = w_{\mathcal{B}_p} + \varepsilon^P$. In addition, the components of $w_{\mathcal{B}}$ are also modified by performing the bound flipping operations. For each index $i \in \{1, \ldots, n\}$ such that $\mathcal{B}_i \in \mathcal{F}$ we have that $\overline{w}_{\mathcal{B}_i} = \text{flip}(w_{\mathcal{B}_i})$. Observe that

$$\text{flip}(w_{\mathcal{B}_i}) = \begin{cases} w_{\mathcal{B}_i} + (u_{\mathcal{B}_i} - l_{\mathcal{B}_i}), & \text{if } \mathcal{B}_i \in \mathcal{B}^l, \\ w_{\mathcal{B}_i} + (l_{\mathcal{B}_i} - u_{\mathcal{B}_i}), & \text{if } \mathcal{B}_i \in \mathcal{B}^u. \end{cases}$$

Hence, after applying the bound flipping and adding the perturbation ε^P, we obtain

$$\begin{aligned} \overline{w}_{\mathcal{B}} &= w_{\mathcal{B}} + \varepsilon^P e_p + \sum_{\substack{i=1 \\ \mathcal{B}_i \in \mathcal{F} \cap \mathcal{B}^l}}^{n} (u_{\mathcal{B}_i} - l_{\mathcal{B}_i}) e_i + \sum_{\substack{i=1 \\ \mathcal{B}_i \in \mathcal{F} \cap \mathcal{B}^u}}^{n} (l_{\mathcal{B}_i} - u_{\mathcal{B}_i}) e_i \\ &= w_{\mathcal{B}} + \varepsilon^P \left(e_p + \sum_{\substack{i=1 \\ \mathcal{B}_i \in \mathcal{F} \cap \mathcal{B}^l}}^{n} \frac{u_{\mathcal{B}_i} - l_{\mathcal{B}_i}}{\varepsilon^P} e_i + \sum_{\substack{i=1 \\ \mathcal{B}_i \in \mathcal{F} \cap \mathcal{B}^u}}^{n} \frac{l_{\mathcal{B}_i} - u_{\mathcal{B}_i}}{\varepsilon^P} e_i \right), \end{aligned} \tag{3.40}$$

where e_i is the i-th row of the identity matrix of order n, $i = 1, \ldots, n$. Notice that the bound flipping corresponds to perturb more than one component of $w_{\mathcal{B}}$. Let $\Delta w_{\mathcal{B}}$ be defined as in (3.39). Then, we can rewrite (3.40) as $\overline{w}_{\mathcal{B}} = w_{\mathcal{B}} + \varepsilon^P \Delta w_{\mathcal{B}}$. This perturbation results in modifying the components of the primal variable x. Indeed, from the primal general solution (3.10), we have that

$$\begin{aligned} \overline{x} &= (B^{-1})^T \left(w_{\mathcal{B}} + \varepsilon^P \Delta w_{\mathcal{B}} \right) \\ &= (B^{-1})^T w_{\mathcal{B}} + \varepsilon^P (B^{-1})^T \Delta w_{\mathcal{B}} \\ &= x + \varepsilon^P \Delta x, \end{aligned} \tag{3.41}$$

where $\Delta x := (B^{-1})^T \Delta w_{\mathcal{B}}$. We also obtain from the primal general solution that

$$\overline{w}_{\mathcal{N}} = N^T (x + \varepsilon^P \Delta x)$$

$$
\begin{aligned}
&= N^T x + \varepsilon^P N^T \Delta x. \\
&= w_{\mathcal{N}} + \varepsilon^P \Delta w_{\mathcal{N}},
\end{aligned}
$$

where $\Delta w_{\mathcal{N}} := N^T \Delta x$. Therefore, we have obtained the expressions of the primal search direction $(\Delta x, \Delta w)$. We still need to show how to compute the perturbation ε^P. Recall that the current s-th nonbasic index enters the basis. Hence, the component $\overline{w}_{\mathcal{N}_s}$ must be equal to one of its bounds after the basis changes. Recall that $\tau = l_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} < l_{\mathcal{N}_s}$, and $\tau = u_{\mathcal{N}_s}$ if $w_{\mathcal{N}_s} > u_{\mathcal{N}_s}$. Thus, in order to obtain $\overline{w}_{\mathcal{N}_s} = \tau$ after the basis change, the perturbation ε^P must satisfy $w_{\mathcal{N}_s} + \varepsilon^P \Delta w_{\mathcal{N}_s} = \tau$. This leads to

$$
\varepsilon^P = \frac{(\tau - w_{\mathcal{N}_s})}{\Delta w_{\mathcal{N}_s}}.
$$

Notice that $\varepsilon^P \neq 0$ and that its computation is well-defined, as $w_{\mathcal{N}_s} = e_s^T w_{\mathcal{N}} = e_s^T N^T x = (a^{\mathcal{N}_s})^T x = (a^{\mathcal{N}_s})^T (B^{-1})^T e_p = -\Delta y_{\mathcal{B}_p}$. By using this perturbation value, the s-th component of $\overline{w}_{\mathcal{N}}$ becomes equal to one of its bound, while the current p-th component of $\overline{w}_{\mathcal{B}}$ may assume any value. Therefore, we have that $(\overline{x}, \overline{w}) := (x, w) + \varepsilon^P (\Delta x, \Delta w)$ is the primal basic solution of the new basis. □

Proposition 3.5.2 proposes a way to update the primal basic solution in order to reflect a basis change. The update is presented in terms of the primal search direction $(\Delta x, \Delta w)$ and the primal step-size ε^P. Alternatively, we can update the components of $w_{\mathcal{B}}$ only, by setting

$$
\overline{w}_{\mathcal{B}_i} := \begin{cases} w_{\mathcal{B}_i} + \varepsilon^P, & \text{if } i = p, \\ \text{flip}(w_{\mathcal{B}_i}), & \text{if } \mathcal{B}_i \in \mathcal{F}, \\ w_{\mathcal{B}_i}, & \text{otherwise,} \end{cases}
$$

for each $i \in \{1, \ldots, n\}$. Then, we compute the remaining components of the new primal basic solution by using (3.10), so that $\overline{x} := (B^{-1})^T w_{\mathcal{B}}$ and $w_{\mathcal{N}} := N^T x$. In practice, this strategy is typically less efficient than using the results of Proposition 3.5.2.

3.6 Computational implementation

In the previous sections, we have presented the theoretical analysis of the dual simplex method for problems in the general form. In addition, we have stated a bound flipping ratio test for this variant and shown how to update the primal and dual basic solutions after performing a basis change. To work well in practice, this method must be implemented by following certain computational techniques that are essential in efficient and stable implementations of a simplex type method (see e.g. Maros, 2003a; Koberstein, 2005; Munari, 2009). In this section, we summarize the main techniques and show how to extend them to the dual simplex method for problems in the general form.

3.6.1 Data structure

Sparsity is a typical characteristic of linear programming formulations which are solved in practice. It happens when the coefficient matrix and the vectors that describe the formulation have relatively few nonzero entries. In general, less than 10% of the entries in the coefficient matrix are different from zero. Hence, we can reduce the amount of memory that is required to store an instance by adopting a sparse representation of its data, in which only the nonzero entries are stored in memory. A sparse representation also reduces the computational effort on

computations, as we avoid operations with zero values. In addition, we can exploit the sparsity of the auxiliary arrays used in the dual simplex method to compute, *e.g.*, the basic solutions and dual search direction. For a review on the main sparse representations used in simplex type methods, see Maros (2003a). In the implementation addressed here, we represent the coefficient matrix of the formulation by both the columnwise and the rowwise representations. Although it may sound redundant, this is a common strategy reported in the literature (Bixby, 2002; Koberstein, 2008), as it allows to efficiently access a matrix either by rows or by columns. These different representations are important, because the performance of some operations are superior when they access the matrix A rows, while others take advantage of columnwise access.

Other operations in the dual simplex method also require carefully designed data structures. For instance, consider the bound flipping ratio test described in Section 3.5. The breakpoints of the dual objective function must be sorted before applying the ratio test. In our implementation, we compute all these values and add them to a priority queue (heap), which keeps them partially sorted (see e.g. Aho et al., 1983). The breakpoints are then iteratively removed from the queue until the conditions of Theorem 3.5.1 are satisfied. We also need special data structures to efficiently solve linear systems of equations. This operation requires an appropriate representation of the basic matrix as well as clever techniques for using this representation to solve linear systems. We address this issue in the next subsection.

3.6.2 Representation of the basic matrix

Currently, efficient implementations of simplex-type methods represent the basic matrix by using the LU factorization and the LU update strategies as proposed by Suhl and Suhl (1990) and Suhl and Suhl (1993). Although the LU factorization is widely used for representing matrices and solving linear systems in many other subjects, the techniques presented by these authors are specific for simplex-type methods, specially the update of the representation after performing basis change. In this section, we extend the Suhl and Suhl techniques to the context of the dual simplex method for problems in the general form.

SSLU Factorization

The LU factorization proposed by Suhl and Suhl (1990), which we call SSLU factorization, consists in transforming the basic matrix B in a matrix $\tilde{U}$, by using $t < n$ elementary matrices $L_1, L_2, \ldots, L_t$, such that

$$L_t L_{t-1} \ldots L_1 B = \tilde{L}^{-1} B = \tilde{U} \quad \Rightarrow \quad B = \tilde{L}\tilde{U}. \tag{3.42}$$

By elementary matrices we mean those that differ from the identity by a single column, and this column has all the entries above the main diagonal equal to zero. For the factorization of B, we may need additional row and column permutation matrices, which we denote respectively by P and Q. These matrices are helpful to express the factors $\tilde{L}$ and $\tilde{U}$ as triangular matrices L and U such that

$$L = P\tilde{L}P^{-1} \;\text{ and }\; U = P\tilde{U}Q. \tag{3.43}$$

With this notation, we have $PBQ = LU$. By convention, L is a lower triangular matrix with unitary diagonal, while U is upper triangular. This nomenclature is extended to $\tilde{L}$ and $\tilde{U}$, which are called the *permuted triangular factors*. $\tilde{L}$ is a permuted lower triangular matrix, and $\tilde{U}$ is a permuted upper triangular matrix.

The position of the columns in the basic matrix is given by the order of the indices in the set $\mathcal{B}$. Hence, the permutation that is determined by matrix Q can be applied directly to $\mathcal{B}$,

instead of being used in the LU factorization. Indeed, if we post-multiply both sides of the equality $\tilde{L}^{-1}B = \tilde{U}$ by the product QP, we obtain

$$\tilde{L}^{-1}B(QP) \;=\; \tilde{U}(QP) \;=\; P^{-1}UQ^{-1}(QP) \;=\; P^{-1}UP.$$

Then, we reorder the columns of B and $\tilde{U}$ using the permutation QP, such that $B := BQP$ and $\tilde{U} := \tilde{U}QP$. As a result, we obtain $\tilde{L}^{-1}B = \tilde{U}$ as before, but now $U = P\tilde{U}P^{-1}$ and therefore matrix Q can be discarded. We say that the triangular factor U is obtained from $\tilde{U}$ by means of *symmetric* permutations, as $P^{-1} = P^T$. All the pivot elements lay on this main diagonal, which is a useful feature for solving linear systems and for updating the representation after a basis change.

SSLU update

After each basis change, a column of the basic matrix must be replaced by the column of the variable that enters the basis. This change requires a new factorization of the basis matrix, which can be computed from scratch. However, to factorize the basic matrix from scratch is a computationally expensive operation in general. By observing this, Suhl and Suhl (1993) proposed a clever strategy for efficiently update it. This strategy has a relatively small computational cost and in addition seeks to maintain the sparsity and the numerical stability of the factorization.

Consider that the basic matrix B is represented by the factors $\tilde{L}^{-1}$ and $\tilde{U}$, which have been obtained by the SSLU factorization. Suppose that, after a basis change, column $a^{\mathcal{N}_s}$ has been chosen to enter the basis at position p, so the new basic matrix is given by $\bar{B} = \left[b_1, \ldots, b_{p-1}, a^{\mathcal{N}_s}, b_{p+1}, \ldots, b_n\right]$. If we premultiply this matrix by $\tilde{L}^{-1}$, then we obtain

$$\begin{aligned}\tilde{L}^{-1}\bar{B} &= \left[\tilde{L}^{-1}b_1, \ldots, \tilde{L}^{-1}b_{p-1}, \tilde{L}^{-1}a^q, \tilde{L}^{-1}b_{p+1}, \ldots, \tilde{L}^{-1}b_n\right] \\ &= \left[\tilde{u}_1, \ldots, \tilde{u}_{p-1}, g, \tilde{u}_{p+1}, \ldots, \tilde{u}_n\right].\end{aligned}$$

Therefore, if we replace the p-th column of $\tilde{U}$ by the column $g = \tilde{L}^{-1}a^{\mathcal{N}_s}$, then the previous permuted triangle factors can be used to represent the new basic matrix. However, since the column g may have nonzero values at any position, we cannot guarantee that the corresponding matrix U remains triangular with this new column. To overcome this, the SSLU update procedure uses an elementary matrix to eliminate the entries of g which are below the pivot entry. Then, by using symmetric permutations of the rows and columns of $\tilde{U}$, the corresponding matrix U recovers the triangular form. The elementary matrix that is used to eliminate entries must be included in the representation of $\tilde{L}^{-1}$. In practice, the implementation should impose a maximum number of elementary matrices that can be used in the representation. When this maximum is achieved, the factorization of the basic matrix must be recomputed from scratch. The refactorization may be motivated by other reasons as well, such as when a maximum number of updates is achieved, or when the numerical instability of the factorization is detected.

Solving linear systems

In the SSLU factorization, we use the factors $\tilde{L}^{-1} = L_t L_{t-1} \ldots L_1$ and $\tilde{U}$ to compute the solution of linear systems involving the basic matrix. The factor $\tilde{L}^{-1}$ is represented as a product of t elementary matrices. The solution α of a linear system of type $B\alpha = \upsilon$ is obtained by an operation that is called *forward transformation* (FTRAN). This operation consists in the following two steps:

$$(i) \quad \text{Compute } v^0 = L_t \dots L_1 v;$$
$$(ii) \quad \text{Solve } \tilde{U}\alpha = v^0.$$

In (ii), the corresponding triangular representation of the factor $\tilde{U}$ should be used. Specifically, the permutation should be applied to the components of v and α prior to the computations. Hence, the solution is given by

$$\tilde{\alpha}_i = \frac{1}{u_{ii}} \left(\tilde{v}_i - \sum_{j=i+1}^{n} u_{ij}\tilde{\alpha}_j \right), \quad i = n, n-1, \dots, 1.$$

where $\tilde{v}$ and $\tilde{\alpha}$ denotes the respective permuted vectors. Observe in this expression that each component $\tilde{\alpha}_j$ multiplies only the column j of U. Therefore, if $\tilde{\alpha}_j = 0$, then the computation may skip column u_j.

We are also interested in solving linear systems of the type $B^T\alpha = v$. In this case, the solution α is obtained by the *backward transformation* (BTRAN)

$$(i) \quad \text{Solve } \tilde{U}^T\alpha^0 = v;$$
$$(ii) \quad \text{Compute } \alpha = L_1^T \dots L_t^T \alpha^0.$$

Step (i) consists in solving a linear system with an upper triangular matrix, which means that the permutation P should be applied to $\tilde{U}^T$. Then, we actually solve the system $U^T\tilde{\alpha}^0 = \tilde{v}$, which has the solution

$$\tilde{\alpha}_j^0 = \frac{1}{u_{jj}} \left(\tilde{v}_j - \sum_{i=1}^{j-1} u_{ij}\tilde{\alpha}_i^0 \right), \quad j = 1, 2, \dots, n.$$

Similarly to what is done in the FTRAN, if $\tilde{\alpha}_i^0 = 0$ then the i-th row of U is skipped in the computations. Finally, the elementary matrices are applied in order to obtain the solution α.

Data structure

Fig. 3.4 illustrates the data structure that we use to store the factors $\tilde{L}^{-1} = L_t L_{t-1} \dots L_1$ and $\tilde{U}$ in our implementation. It follows the discussions presented in Suhl and Suhl (1990, 1993); Koberstein (2005). The permuted triangular factor $\tilde{U}$ is stored using both columnwise and rowwise forms. For each entry in $\tilde{U}$, there is a pointer linking its position in the rowwise representation to its position in the columnwise representation. By using these pointers, we avoid searching for a given entry of the matrix, a operation that would slow down the factorization and updating procedures. The gray areas in the figure represent free regions which are used to store nonzero entries that are created during the SSLU update. The elementary matrices $L_t L_{t-1} \dots L_1$ are stored in the last positions of the rowwise representation. Only the entries in nontrivial columns/rows of these matrices are stored.

3.6.3 Scaling

Scaling is an essential operation to avoid numerical instability in a simplex-type method. It consists in modifying the coefficient matrix of the problem in order to obtain a matrix with better numerical characteristics. The modification involves multiplying rows and columns by scaling factors, without modifying the set of optimal solutions of the problem. Many scaling techniques are proposed in the literature, and they differ basically in how to compute

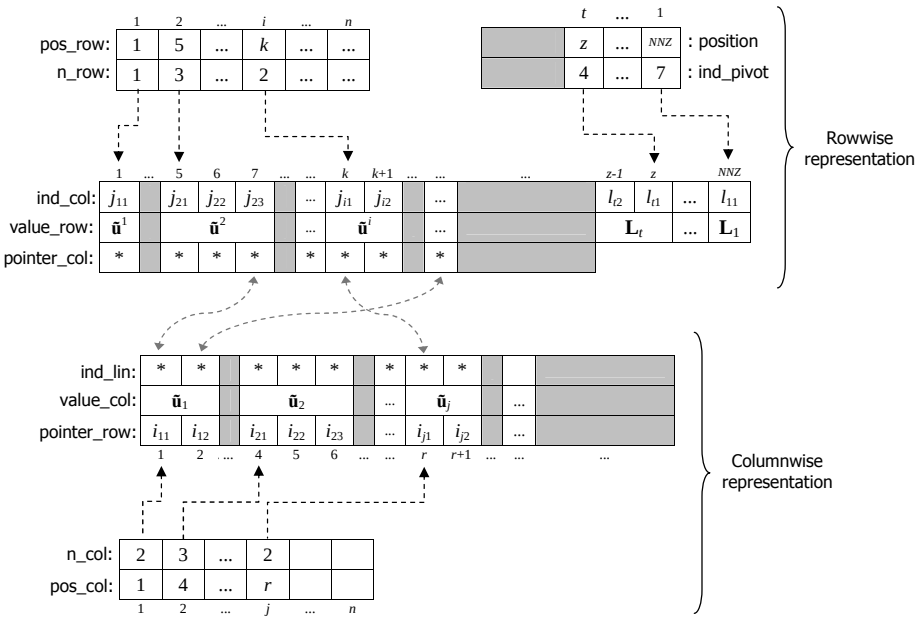

Figure 3.4: Data structure that stores the permuted triangular factors.

the scaling factors. Munari and Arenales (2009) present a review of the scaling techniques that are most used in implementations of simplex-type methods. In addition, the authors compare the performance of the techniques when solving the Netlib instances by simplex-type methods. According to the reported results, the geometric average scaling and the scaling technique proposed by Benichou et al. (1977) are the best strategies concerning CPU time and numerical stability. In the implementation of the dual simplex method for problems in the general form, we have tested all the scaling techniques reviewed by Munari and Arenales (2009). Among them, the geometric average scaling led to the best overall performance of the method. Hence, this technique has been adopted as default in our implementation.

3.6.4 Numerical tolerances

Approximations are inevitable when making computations with fractional values in a computer. Due to the floating point representations, the approximations may happen in the assignment of values to variables as well as during calculations. Based on this, we should be careful when comparing two given values, and use a tolerance in the comparisons. For instance, to verify if the values a and b are equal, we should actually verify if $|a - b| < \delta$, where δ is a tolerance which guarantees that the values are sufficiently close to each other. In general, the appropriate choice of δ depends on the type of comparison and the magnitude of the values. The different tolerance values used in our implementation are described in Table 3.1, according to their usage.

3.7 Results and discussion

In this section, we analyze the practical performance of the dual simplex method for problems in the general form (DSMGF). The experiments are based on benchmarking instances from the Netlib, a public repository of linear programming problems. First, we compare two variants of the DSMGF in order to verify the impact of the bound flipping ratio test in relation to the standard ratio test. Then, the variant with the best overall performance is compared

Tolerance	Usage
10^{-6}	minimum absolute value allowed for the pivot
10^{-1}	minimum relative value allows for the pivot
10^{-6}	primal feasibility test (*pricing* operation)
10^{-9}	dual feasibility test (ratio test)
10^{-10}	zero tolerance
10^{-14}	drop tolerance (reset the value as zero)

Table 3.1: Numerical tolerances used in our computational implementation.

against other standard simplex-type methods. We have implemented all the methods in the C programming language, including the procedures to factorize and update the basic matrix. The experiments were performed on a Linux PC with an Intel Core i7 2.8 GHz CPU and 8.0 GB of memory.

The Netlib is a repository of linear programming problems, which is widely used to benchmark simplex-type methods. Many instances in this set correspond to real-life problems that are very challenging to simplex-type methods. To solve these problems efficiently, it is important to exploit the sparsity of the data and to use techniques for controlling the numerical stability of the method. The instances are publicly available at `http://www.netlib.org/lp/data`. Tables 3.2 and 3.3 summarize the main information regarding the instances. Columns n and $\overline{m}$ give the number of variables and the number of nontrivial constraints in the instance, respectively. The remaining columns are the number of variables that are free (f), that have only one finite bound (p), and that are boxed (b), considering the problem is formulated in the general form. Similar columns are presented to describe the types of constraints. The number of fixed variables are shown inside parenthesis in column (b). This value is omitted in case it is zero. The density of the coefficient matrix (d) and the optimal value of the instance are given in the last two columns. For further information regarding these instances, please see Koch (2004); Gay (1997).

3.7.1 Checking the impact of the bound flipping ratio test

In the first set of computational experiments, we verify the importance of using the bound flipping ratio test in the DSMGF. We have solved the Netlib instances by using two variants of the DSMGF: one using the standard ratio test and another using the bound flipping ratio test. We refer to these variants as DSMGF-SRT and DSMGF-BFRT, respectively. Tables 3.4 and 3.5 show the number of iterations and the CPU time (in seconds) to solve the Netlib instances by using the two variants of the DSMGF. The total number of iterations and the total CPU time to solve all the instances are presented in the last row of Table 3.5. According to the results, the bound flipping ratio test was helpful to reduce the number of iterations to solve the instances. Moreover, the variant using the bound flipping ratio test was able to solve all instances, while the other variant failed in 3 instances (due to numerical difficulties). In the vast majority of instances, the variant with the standard ratio test (DSMGF-SRT) presented a larger number of iterations and, in total, it required 44% more iterations in relation to the DSMGF-BFRT. The largest impact of the bound flipping ratio test is observed for the instances FIT2D, FIT1D, PDS-06, KEN-18. These are instances in which most of the variables and constraints are boxed, which benefits the bound flipping (see Theorem 3.5.1). The bound flipping ratio test also contributed to reduce the CPU time in many instances. In total, the DSMGF-BFRT was 2.15 times faster than the other variant. Therefore, the results

Problem	n	$\overline{m}$	Variables			Constraints		d (%)	Optimal value
			f	p	b	p	b		
25FV47	1571	821	0	1571	0	305	516	0.806	5.5018458883E+03
80BAU3B	9799	2262	0	6315	3484 (498)	2262	0	0.095	9.8722419241E+05
ADLITTLE	97	56	0	97	0	41	15	7.051	2.2549496316E+05
AFIRO	32	27	0	32	0	19	8	9.606	-4.6475314286E+02
AGG	163	488	0	163	0	452	36	3.030	-3.5991767287E+07
AGG2	302	516	0	302	0	456	60	2.749	-2.0239252356E+07
AGG3	302	516	0	302	0	456	60	2.759	1.0312115935E+07
BANDM	472	305	0	472	0	0	305	1.732	-1.5862801845E+02
BEACONFD	262	173	0	262	0	33	140	7.446	3.3592485807E+04
BLEND	83	74	0	83	0	31	43	7.994	-3.0812149846E+01
BNL1	1175	643	0	1175	0	411	232	0.678	1.9776295615E+03
BNL2	3489	2324	0	3489	0	997	1327	0.173	1.8112365404E+03
BOEING1	384	351	0	228	156	253	98	2.593	-3.3521356751E+02
BOEING2	143	166	0	89	54	143	23	5.038	-3.1501872802E+02
BORE3D	315	233	0	303	12 (1)	19	214	1.947	1.3730803942E+03
BRANDY	249	220	0	249	0	54	166	3.921	1.5185098965E+03
CAPRI	353	271	14	192	147 (16)	129	142	1.847	2.6900129138E+03
CRE-A	4067	3516	0	4067	0	3181	335	0.105	2.3595407061E+07
CRE-B	72447	9648	0	72447	0	4690	4958	0.037	2.3129639887E+07
CRE-C	3678	3068	0	3678	0	2733	335	0.117	2.5275116141E+07
CRE-D	69980	8926	0	69980	0	3968	4958	0.039	2.4454969765E+07
CYCLE	2857	1903	7	2773	77	514	1389	0.381	-5.2263930249E+00
CZPROB	3523	929	0	3294	229 (229)	39	890	0.326	2.1851966989E+06
D2Q06C	5167	2171	0	5167	0	664	1507	0.289	1.2278421081E+05
D6CUBE	6184	415	0	6184	0	0	415	1.469	3.1549166667E+02
DEGEN2	534	444	0	534	0	223	221	1.678	-1.4351780000E+03
DEGEN3	1818	1503	0	1818	0	786	717	0.902	-9.8729400000E+02
E226	282	223	0	282	0	190	33	4.099	-1.8751929066E+01
ETAMACRO	688	400	0	471	217 (82)	128	272	0.875	-7.5571523337E+02
FFFFF800	854	524	0	854	0	174	350	1.392	5.5567956482E+05
FINNIS	614	497	0	533	81 (45)	450	47	0.757	1.7279106560E+05
FIT1D	1026	24	0	0	1026	23	1	54.435	-9.1463780924E+03
FIT1P	1677	627	0	1278	399	0	627	0.938	9.1463780924E+03
FIT2D	10500	25	0	0	10500	24	1	49.150	-6.8464293294E+04
FIT2P	13525	3000	0	6025	7500	0	3000	0.124	6.8464293294E+04
FORPLAN	421	161	0	397	24 (3)	70	91	6.732	-6.6421896127E+02
GANGES	1681	1309	0	1284	397	25	1284	0.314	-1.0958573613E+05
GFRD-PNC	1092	616	0	834	258	68	548	0.353	6.9022359995E+06
GREENBEA	5405	2392	0	5012	393 (103)	193	2199	0.239	-7.2555248130E+07
GREENBEB	5405	2392	4	4995	406 (115)	193	2199	0.239	-4.3022602612E+06
GROW15	645	300	0	45	600	0	300	2.904	-1.0687094129E+08
GROW22	946	440	0	66	880	0	440	1.983	-1.6083433648E+08
GROW7	301	140	0	21	280	0	140	6.198	-4.7787811815E+07
ISRAEL	142	174	0	142	0	174	0	9.183	-8.9664482186E+05
KB2	41	43	0	32	9	27	16	16.222	-1.7499001299E+03
KEN-07	3602	2426	0	0	3602	0	2426	0.096	-6.7952044338E+08
KEN-11	21349	14694	0	0	21349	0	14694	0.016	-6.9723822625E+09
KEN-13	42659	28632	0	0	42659	0	28632	0.008	-1.0257394789E+10
KEN-18	154699	105127	0	0	154699	0	105127	0.002	-5.2217025287E+10
LOTFI	308	153	0	308	0	58	95	2.288	-2.5264706062E+01
MAROS	1443	846	0	1408	35	523	323	0.788	-5.8063743701E+04
MAROS-R7	9408	3136	0	9408	0	0	3136	0.491	1.4971851665E+06
MODSZK1	1620	687	2	1618	0	0	687	0.285	3.2061972906E+02
NESM	2923	662	0	1240	1683 (175)	94	568	0.687	1.4076036488E+07
OSA-07	23949	1118	0	23949	0	1118	0	0.537	5.3572251730E+05
OSA-14	52460	2337	0	52460	0	2337	0	0.257	1.1064628447E+06

Table 3.2: Description of the Netlib instances. (Part 1)

Problem	n	$\overline{m}$	Variables			Constraints		d (%)	Optimal value
			f	p	b	p	b		
OSA-30	100024	4350	0	100024	0	4350	0	0.138	2.1421398732E+06
OSA-60	232966	10280	0	232966	0	10280	0	0.058	4.0440725032E+06
PDS-02	7535	2953	0	5401	2134	181	2772	0.074	2.8857862010E+10
PDS-06	28655	9881	0	19415	9240	696	9185	0.022	2.7761037600E+10
PDS-10	48763	16558	0	32615	16148	1169	15389	0.013	2.6727094976E+10
PDS-20	105728	33874	0	70840	34888	2447	31427	0.006	2.3821658640E+10
PEROLD	1376	625	88	958	330 (64)	130	495	0.700	-9.3807552782E+03
PILOT	3652	1441	0	2409	1243 (167)	1208	233	0.820	-5.5748972928E+02
PILOT.JA	1988	940	88	1250	650 (311)	279	661	0.787	-6.1131364656E+03
PILOT.WE	2789	722	80	2337	372 (78)	139	583	0.453	-2.7201075328E+06
PILOT4	1000	410	88	635	277 (30)	123	287	1.254	-2.5811392589E+03
PILOT87	4883	2030	0	3085	1798 (180)	1797	233	0.738	3.0171034733E+02
PILOTNOV	2172	975	0	1628	544 (204)	274	701	0.617	-4.4972761882E+03
RECIPE	180	91	0	85	95 (24)	24	67	4.048	-2.6661600000E+02
SC105	103	105	0	103	0	60	45	2.589	-5.2202061212E+01
SC205	203	205	0	203	0	114	91	1.324	-5.2202061212E+01
SC50A	48	50	0	48	0	30	20	5.417	-6.4575077059E+01
SC50B	48	50	0	48	0	30	20	4.917	-7.0000000000E+01
SCAGR25	500	471	0	500	0	171	300	0.660	-1.4753433061E+07
SCAGR7	140	129	0	140	0	45	84	2.326	-2.3313898243E+06
SCFXM1	457	330	0	457	0	143	187	1.717	1.8416759028E+04
SCFXM2	914	660	0	914	0	286	374	0.859	3.6660261565E+04
SCFXM3	1371	990	0	1371	0	429	561	0.573	5.4901254550E+04
SCORPION	358	388	0	358	0	108	280	1.027	1.8781248227E+03
SCRS8	1169	490	0	1169	0	106	384	0.556	9.0429695380E+02
SCSD1	760	77	0	760	0	0	77	4.081	8.6666666743E+00
SCSD6	1350	147	0	1350	0	0	147	2.175	5.0500000077E+01
SCSD8	2750	397	0	2750	0	0	397	0.786	9.0499999993E+02
SCTAP1	480	300	0	480	0	180	120	1.175	1.4122500000E+03
SCTAP2	1880	1090	0	1880	0	620	470	0.328	1.7248071429E+03
SCTAP3	2480	1480	0	2480	0	860	620	0.242	1.4240000000E+03
SEBA	1028	515	0	521	507	1	514	0.822	1.5711600000E+04
SHARE1B	225	117	0	225	0	28	89	4.372	-7.6589318579E+04
SHARE2B	79	96	0	79	0	83	13	9.151	-4.1573224074E+02
SHELL	1775	536	0	1408	367 (250)	2	534	0.374	1.2088253460E+09
SHIP04L	2118	402	0	2118	0	48	354	0.744	1.7933245380E+06
SHIP04S	1458	402	0	1458	0	48	354	0.743	1.7987147004E+06
SHIP08L	4283	778	0	4283	0	80	698	0.384	1.9090552114E+06
SHIP08S	2387	778	0	2387	0	80	698	0.383	1.9200982105E+06
SHIP12L	5427	1151	0	5427	0	106	1045	0.259	1.4701879193E+06
SHIP12S	2763	1151	0	2763	0	106	1045	0.257	1.4892361344E+06
SIERRA	2036	1227	0	0	2036	699	528	0.292	1.5394362184E+07
STAIR	467	356	6	373	88 (82)	147	209	2.319	-2.5126695119E+02
STANDATA	1075	359	0	955	120 (16)	199	160	0.785	1.2576995000E+03
STANDMPS	1075	467	0	955	120 (16)	199	268	0.733	1.4060175000E+03
STOCFOR1	111	117	0	111	0	54	63	3.442	-4.1131976219E+04
STOCFOR2	2031	2157	0	2031	0	1014	1143	0.190	-3.9024408538E+04
STOCFOR3	15695	16675	0	15695	0	7846	8829	0.030	-3.9976783944E+04
TRUSS	8806	1000	0	8806	0	0	1000	0.316	4.5881584719E+05
TUFF	587	333	2	556	29 (3)	41	292	2.312	2.9214776509E-01
VTP.BASE	203	198	1	119	83 (18)	143	55	2.259	1.2983146246E+05
WOOD1P	2594	244	0	2594	0	1	243	11.094	1.4429024116E+00
WOODW	8405	1098	0	8405	0	13	1085	0.406	1.3044763331E+00

Table 3.3: Description of the Netlib instances. (Part 2)

indicate that the DSMGF-BFRT outperforms the DSMGF-SRT. For this reason, we adopt the bound flipping ratio test as default in the experiments we report in the next sections.

3.7.2 Comparing against standard simplex-type methods

The second set of experiments consisted in comparing the performance of the DSMGF (with bound flipping ratio test) against the performance of standard simplex-type methods, namely the primal simplex method for problems in the standard form (PSMSF) and the dual simplex method for problems in the standard form (DSMSF). By standard form we refer to a formulation in which the constraints are modeled by a set of equations (see Chapter 2). Before solving a problem in the standard, we usually need to the transform inequalities and boxed constraints into equalities, by means of slack or surplus variables.

We have implemented both the PSMSF and the DSMSF, following the descriptions in Munari (2009) and Maros (2003a). The implementation has the same (or very similar) data structures and computational techniques as those used in the implementation of the DSMGF. Hence, all the methods rely on the same implementation of core components and we have specialized the code according to each variant. This allows us to verify the performance of the different variants without having the results biased to the level of the implementation skills. It is worth mentioning that in our implementation of the DSMSF, we also use the bound flipping ratio test as default.

Tables 3.4 and 3.5 show the number of iterations and the CPU time in seconds to solve the Netlib instances by the three variants. First, we observe that the PSMSF is clearly outperformed by the other two variants. A strong reason for that is the use of the bound flipping ratio test in the dual variants, as no equivalent strategy is available for the primal method. Indeed, dual simplex-type methods have been recognized as superior in the literature (Bixby, 2002; Maros, 2003b; Koberstein, 2008). A very similar performance is observed for the two dual variants, the DSMSF and the DSMGF, with the latter presenting slightly better results. The DSMGF was the fastest variant in many difficult instances, such as CRE-B, PDS-10 and KEN-18. In total, the DSMGF was 10% faster than the DSMSF.

3.8 Concluding remarks

In this chapter, we addressed a variant of the dual simplex method which exploits special properties of the general form. First, we presented a novel theoretical discussion of the variant, which extends the preliminary studies of Arenales (1984), Sousa et al. (2005) and Silva et al. (2007). Then, we discussed the main techniques that are useful to obtain an efficient and stable implementation of the method. Finally, we presented computational experiments with the Netlib instances, which are widely used to benchmark simplex-type methods. The results indicate that the dual simplex method for problems in the general form is competitive with the standard dual simplex method and outperforms the standard primal simplex method. In particular, the proposed method has the best relative performance in many of the most difficult instances, namely CRE-B, PDS-10 and KEN-18.

Further topics can be investigated regarding the dual simplex method for problems in the general form. First, the performance of the method should be verified for larger instances, specially those corresponding to relaxations of integer programming problems. Also, it would be interesting to see the behavior of the method within integer programming methodologies, such as the branch-and-bound method and the cutting plane method. Furthermore, it is important to investigate strategies for obtaining advanced basis to initialize the method.

	DSMGF-SRT		DSMGF-BFRT	
Instance	Iterations	Time (in s)	Iterations	Time (in s)
25FV47	4568	0.62	4656	0.64
80BAU3B	7992	2.01	4464	0.79
ADLITTLE	136	0.01	150	0.00
AFIRO	32	0.00	37	0.00
AGG	323	0.03	322	0.05
AGG2	375	0.02	385	0.03
AGG3	382	0.04	392	0.03
BANDM	700	0.07	648	0.07
BEACONFD	111	0.01	111	0.02
BLEND	118	0.01	136	0.00
BNL1	1661	0.20	1796	0.24
BNL2	3057	0.43	3105	0.44
BOEING1	1018	0.11	662	0.08
BOEING2	261	0.03	254	0.01
BORE3D	249	0.02	260	0.03
BRANDY	453	0.04	474	0.06
CAPRI	416	0.03	307	0.03
CRE-A	2900	0.44	3040	0.44
CRE-B	40243	85.09	43655	86.88
CRE-C	2910	0.46	2800	0.40
CRE-D	39164	77.39	38537	69.03
CYCLE	3841	0.55	3201	0.46
CZPROB	3156	0.44	3121	0.44
D2Q06C	22764	6.03	21994	4.61
D6CUBE	1118	0.26	1186	0.27
DEGEN2	2497	0.27	2386	0.25
DEGEN3	12554	1.87	12247	1.76
E226	490	0.05	440	0.05
ETAMACRO	1082	0.12	832	0.10
FFFFF800	852	0.08	1083	0.12
FINNIS	461	0.05	436	0.05
FIT1D	612	0.07	85	0.01
FIT1P	3931	0.53	3685	0.51
FIT2D	6951	3.03	207	0.11
FIT2P	48748	54.14	31659	29.95
FORPLAN	276	0.01	273	0.02
GANGES	1468	0.20	1326	0.17
GFRD-PNC	644	0.08	541	0.07
GREENBEA	14290	3.75	13237	2.92
GREENBEB	14708	3.51	15779	3.30
GROW15	763	0.07	3033	0.39
GROW22	1748	0.23	4691	0.65
GROW7	394	0.04	726	0.09
ISRAEL	197	0.02	175	0.00
KB2	53	0.00	43	0.02
KEN-07	5237	0.78	3189	0.39
KEN-11	34608	32.65	20379	15.08
KEN-13	108475	242.98	57089	111.07
KEN-18	472306	4619.39	193303	1847.87
LOTFI	389	0.03	536	0.04
MAROS	2669	0.36	2839	0.39
MAROS-R7	4934	5.64	4980	5.33
MODSZK1	—	—	679	0.08
NESM	3413	0.47	1910	0.25
OSA-07	1048	0.76	1048	1.34
OSA-14	2511	4.26	2503	4.48

Table 3.4: Number of iterations and CPU time (in seconds) to solve the Netlib instances by using two variants of the dual simplex method for problems in the general form: one using the standard ration test (DSMGF-SRT) and another using the bound flipping ratio test (DSMGF-BFRT). (Part 1)

	DSMGF (SRT)		DSMGF (BFRT)	
Instance	Iterations	Time (in s)	Iterations	Time (in s)
OSA-30	6136	20.62	6109	20.11
OSA-60	14404	128.85	14295	116.18
PDS-02	3790	0.74	3027	0.53
PDS-06	76873	73.75	21584	15.39
PDS-10	—	—	85773	128.67
PEROLD	3868	0.53	3115	0.43
PILOT	12570	4.08	8697	2.62
PILOT.JA	4012	0.57	2953	0.41
PILOT.WE	8766	1.25	6974	0.99
PILOT4	2774	0.36	1182	0.17
PILOT87	—	—	13544	12.89
PILOTNOV	1590	0.20	1425	0.19
RECIPE	45	0.00	45	0.00
SC105	98	0.00	92	0.00
SC205	206	0.03	212	0.03
SC50A	46	0.00	45	0.02
SC50B	49	0.00	49	0.00
SCAGR25	723	0.09	729	0.07
SCAGR7	189	0.03	189	0.03
SCFXM1	402	0.03	410	0.03
SCFXM2	915	0.09	907	0.10
SCFXM3	1409	0.19	1418	0.17
SCORPION	356	0.03	356	0.03
SCRS8	816	0.12	824	0.12
SCSD1	115	0.01	114	0.01
SCSD6	443	0.05	479	0.05
SCSD8	2123	0.28	4147	0.59
SCTAP1	238	0.02	301	0.01
SCTAP2	695	0.09	766	0.09
SCTAP3	944	0.11	1046	0.11
SEBA	441	0.05	440	0.05
SHARE1B	258	0.01	267	0.03
SHARE2B	124	0.00	122	0.00
SHELL	754	0.08	766	0.09
SHIP04L	438	0.05	439	0.04
SHIP04S	435	0.05	437	0.05
SHIP08L	831	0.10	830	0.11
SHIP08S	771	0.09	771	0.09
SHIP12L	1304	0.20	1305	0.17
SHIP12S	1183	0.14	1187	0.13
SIERRA	706	0.08	671	0.07
STAIR	482	0.06	488	0.06
STANDATA	78	0.00	79	0.00
STANDMPS	224	0.03	226	0.01
STOCFOR1	109	0.00	123	0.01
STOCFOR2	2485	0.34	2551	0.34
STOCFOR3	23391	17.09	23854	15.50
TRUSS	10507	3.08	11136	3.13
TUFF	301	0.03	296	0.01
VTP.BASE	224	0.03	201	0.04
WOOD1P	275	0.06	278	0.06
WOODW	1647	0.40	1685	0.37
Total*	1077820	5403.80	745991	2512.25

* It also adds the results in the Table 3.4.

Table 3.5: Number of iterations and CPU time (in seconds) to solve the Netlib instances by using two variants of the dual simplex method for problems in the general form: one using the standard ration test (DSMGF-SRT) and another using the bound flipping ratio test (DSMGF-BFRT). (Part 2)

	PSMSF		DSMSF		DSMGF	
Instance	Iterations	Time (in s)	Iterations	Time (in s)	Iterations	Time (in s)
25FV47	6605	0.92	4962	0.69	4656	0.64
80BAU3B	9447	1.87	4306	0.86	4464	0.79
ADLITTLE	134	0.00	141	0.01	150	0.00
AFIRO	23	0.00	37	0.00	37	0.00
AGG	122	0.02	194	0.02	322	0.05
AGG2	193	0.03	277	0.02	385	0.03
AGG3	174	0.01	279	0.03	392	0.03
BANDM	591	0.06	605	0.06	648	0.07
BEACONFD	112	0.00	111	0.00	111	0.02
BLEND	95	0.01	133	0.01	136	0.00
BNL1	2239	0.28	1872	0.22	1796	0.24
BNL2	6720	1.58	2998	0.43	3105	0.44
BOEING1	680	0.10	579	0.06	662	0.08
BOEING2	174	0.02	249	0.02	254	0.01
BORE3D	211	0.03	204	0.01	260	0.03
BRANDY	361	0.06	515	0.06	474	0.06
CAPRI	509	0.08	308	0.03	307	0.03
CRE-A	4164	1.14	2830	0.40	3040	0.44
CRE-B	261002	776.97	40420	115.33	43655	86.88
CRE-C	4844	0.77	2913	0.41	2800	0.40
CRE-D	63757	160.39	30928	73.14	38537	69.03
CYCLE	1121	0.33	2935	0.42	3201	0.46
CZPROB	1965	0.26	1688	0.24	3121	0.44
D2Q06C	33194	10.86	22293	6.15	21994	4.61
D6CUBE	204300	61.25	1022	0.27	1186	0.27
DEGEN2	4665	0.52	1943	0.21	2386	0.25
DEGEN3	194262	33.73	10175	1.50	12247	1.76
E226	561	0.06	475	0.05	440	0.05
ETAMACRO	720	0.06	870	0.08	832	0.10
FFFFF800	990	0.12	981	0.11	1083	0.12
FINNIS	690	0.08	426	0.04	436	0.05
FIT1D	1058	0.12	85	0.01	85	0.01
FIT1P	4165	0.57	3685	0.48	3685	0.51
FIT2D	11500	6.99	260	0.15	207	0.11
FIT2P	92356	56.02	31919	12.93	31659	29.95
FORPLAN	264	0.01	274	0.01	273	0.02
GANGES	1613	0.17	1305	0.16	1326	0.17
GFRD-PNC	1065	0.11	541	0.06	541	0.07
GREENBEA	17265	4.18	10091	2.41	13237	2.92
GREENBEB	13341	3.18	17706	4.11	15779	3.30
GROW15	791	0.10	2614	0.32	3033	0.39
GROW22	1256	0.17	5670	0.77	4691	0.65
GROW7	330	0.02	714	0.09	726	0.09
ISRAEL	161	0.03	168	0.01	175	0.00
KB2	71	0.00	43	0.00	43	0.02
KEN-07	4320	0.62	3177	0.47	3189	0.39
KEN-11	30461	31.12	20352	19.72	20379	15.08
KEN-13	88288	216.14	55838	133.23	57089	111.07
KEN-18	361040	4495.67	199352	2058.71	193303	1847.87
LOTFI	286	0.02	493	0.06	536	0.04
MAROS	1664	0.20	2571	0.34	2839	0.39
MAROS-R7	3897	2.74	4995	6.03	4980	5.33
MODSZK1	1085	0.12	688	0.06	679	0.08
NESM	4465	0.58	2127	0.27	1910	0.25
OSA-07	804	0.39	1047	1.17	1048	1.34
OSA-14	1666	1.89	2505	3.45	2503	4.48

Table 3.6: Number of iterations and CPU time (in seconds) to solve the Netlib instances by using three simplex type methods: the primal simplex method for problems in the standard form (PSMSF), the dual simplex method for problems in the standard form (DSMSF), and the dual simplex method for problems in the general form (DSMGF). (Part 1)

	PSMSF		DSMSF		DSMGF	
Instance	Iterations	Time (in s)	Iterations	Time (in s)	Iterations	Time (in s)
OSA-30	3274	7.33	6087	16.28	6109	20.11
OSA-60	7278	38.49	14878	94.39	14295	116.18
PDS-02	8196	1.65	3067	0.49	3027	0.53
PDS-06	83266	78.06	23911	15.83	21584	15.39
PDS-10	172734	334.97	107472	160.31	85773	128.67
PEROLD	6267	0.86	2957	0.39	3115	0.43
PILOT	11856	5.71	8367	2.81	8697	2.62
PILOT.JA	7437	1.05	3209	0.45	2953	0.41
PILOT.WE	9063	1.27	6538	0.95	6974	0.99
PILOT4	1289	0.16	1250	0.14	1182	0.17
PILOT87	19791	17.64	12130	11.65	13544	12.89
PILOTNOV	2760	0.38	1402	0.18	1425	0.19
RECIPE	51	0.00	45	0.00	45	0.00
SC105	114	0.02	92	0.00	92	0.00
SC205	245	0.02	202	0.02	212	0.03
SC50A	50	0.01	45	0.00	45	0.02
SC50B	50	0.00	49	0.00	49	0.00
SCAGR25	585	0.05	688	0.06	729	0.07
SCAGR7	165	0.01	177	0.02	189	0.03
SCFXM1	393	0.05	406	0.03	410	0.03
SCFXM2	794	0.07	903	0.10	907	0.10
SCFXM3	1187	0.14	1402	0.17	1418	0.17
SCORPION	377	0.05	347	0.05	356	0.03
SCRS8	686	0.07	592	0.06	824	0.12
SCSD1	304	0.02	109	0.00	114	0.01
SCSD6	784	0.08	456	0.04	479	0.05
SCSD8	1940	0.25	2894	0.39	4147	0.59
SCTAP1	266	0.02	310	0.04	301	0.01
SCTAP2	902	0.11	745	0.07	766	0.09
SCTAP3	1232	0.17	1047	0.12	1046	0.11
SEBA	501	0.06	440	0.05	440	0.05
SHARE1B	283	0.02	265	0.03	267	0.03
SHARE2B	131	0.00	121	0.00	122	0.00
SHELL	814	0.08	759	0.07	766	0.09
SHIP04L	583	0.05	414	0.02	439	0.04
SHIP04S	510	0.05	418	0.03	437	0.05
SHIP08L	1078	0.14	810	0.10	830	0.11
SHIP08S	832	0.10	732	0.09	771	0.09
SHIP12L	1564	0.21	1288	0.17	1305	0.17
SHIP12S	1194	0.16	1185	0.15	1187	0.13
SIERRA	1062	0.13	670	0.07	671	0.07
STAIR	839	0.11	482	0.05	488	0.06
STANDATA	71	0.00	73	0.00	79	0.00
STANDMPS	380	0.04	211	0.01	226	0.01
STOCFOR1	117	0.00	118	0.02	123	0.01
STOCFOR2	2883	0.40	2402	0.32	2551	0.34
STOCFOR3	35221	33.24	21395	13.39	23854	15.50
TRUSS	10181	2.59	9025	2.69	11136	3.13
TUFF	372	0.02	334	0.03	296	0.01
VTP.BASE	215	0.03	198	0.02	201	0.04
WOOD1P	334	0.08	278	0.05	278	0.06
WOODW	2019	0.37	1683	0.37	1685	0.37
Total*	1852357	6399.31	750967	2768.65	745991	2512.25

* It also adds the results in the Table 3.6.

Table 3.7: Number of iterations and CPU time (in seconds) to solve the Netlib instances by using three simplex type methods: the primal simplex method for problems in the standard form (PSMSF), the dual simplex method for problems in the standard form (DSMSF), and the dual simplex method for problems in the general form (DSMGF). (Part 2)

Chapter 4

Using the primal-dual interior point algorithm within the column generation method

* A compact version of this chapter has been published in the *European Journal of Operational Research* (see Gondzio et al., 2013).

Column generation plays an important role in optimization. This approach has been applied with success to problems that arise in several contexts, such as integer programming (Lübbecke and Desrosiers, 2005; Vanderbeck and Wolsey, 2010), nonlinear programming and non-differentiable optimization (Goffin and Vial, 2002). For instance, integer programming methodologies that are based on the column generation method are nowadays the most efficient exact approaches for solving real-life problems such as the cutting stock problem and the vehicle routing problem (Degraeve and Peeters, 2003; Baldacci et al., 2012).

The column generation method is an iterative procedure that is used to solve a linear programming problem in which the number of variables is huge, typically exponential, and the columns in the coefficient matrix of this problem can be generated by following a given rule. The method starts with an auxiliary problem which has only a subset of the variables (columns) of the original linear programming problem. Each iteration of the method consists in two basic steps: (*i*) solving the auxiliary problem to obtain a dual solution; (*ii*) using the obtained dual solution to generate one or more columns that are added to the auxiliary problem. The method terminates when the optimal solution of the auxiliary problem is guaranteed to be an optimal solution of the original problem. A full description of the column generation method is given in Section 4.1 of this chapter.

The dual solutions used to generate columns play a fundamental role in the performance of the column generation method. The standard strategy is to use an optimal dual solution of the auxiliary problem. However, as pointed out in the literature, optimal dual solutions typically cause an unstable behavior of the method, specially when they are extreme points of the auxiliary problem. The reason is because these solutions are likely to oscillate sharply from one iteration to another (see Section 4.2 for an illustration), slowing down the progress of the method. Due to this behavior, strategies that are based on stable dual solutions have been proposed in the literature, leading to more efficient variants of the column generation method. We review most of these variants in Section 4.2.

In this chapter, we address a variant of the column generation method in which the primal-dual interior point method is used to solve the auxiliary problem at each iteration. In this

variant, we exploit an important advantage provided by the interior point method, namely the use of well-centered solutions. The centrality comes at no extra cost, as it is a standard feature of interior point methods. Furthermore, by combining this feature with early termination (*i.e.*, the use of a loose optimality tolerance), the obtained dual solutions are likely to be stable and lead to a good overall performance of the column generation method. The proposed method is based on the preliminary investigations presented by Gondzio and Sarkissian (1996) and Martinson and Tind (1999). In Section 4.3, we give a complete description of the method and present a novel theoretical analysis regarding its convergence. In Section 4.4, we present an extensive computational study of the primal-dual column generation method based on linear relaxations obtained from classical integer programming problems, namely the cutting stock problem, the vehicle routing problem with time windows, and the capacitated lot sizing problem with setup times.

It is worth mentioning that we are not concerned with obtaining optimal integer solutions in the study addressed in this chapter. This would require the combination of the column generation method with a branch-and-bound tree, which is known as the branch-and-price method. Instead, we want to analyze the behavior of the proposed approach in a given node of the branch-and-price tree. By improving the performance of the column generation procedure, we are likely to improve the overall performance of solving the integer problem to optimality. The use of the primal-dual interior point method within a branch-and-price methodology will be addressed in Chapter 5.

4.1 The column generation method

In this section, we describe the fundamental concepts of the standard column generation method. The method was first used by Ford and Fulkerson (1958) as a generalization of the simplex method. The motivation came from the need of treating the nonbasic variables implicitly in the pricing operation of the simplex method, because the number of variables was too big to be dealt with explicitly. At that time, the same strategy was successfully used by other authors (Dantzig and Wolfe, 1960; Gilmore and Gomory, 1961, 1963). Simultaneously, Kelley (1960) proposed the cutting plane method, a general-purpose method for optimizing a nonlinear function by using first order approximations of the objective function. The cutting plane method is the same as the column generation method applied to the dual problem. For this reason, the terms *column generation method* and *cutting plane method* are often used as synonyms.

In the column generation literature, the linear programming problem we want to solve is called the *master problem* (MP), which is represented here by

$$z^\star := \min \quad \sum_{j \in N} c_j \lambda_j, \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{j \in N} a_j \lambda_j = b, \tag{4.1b}$$

$$\lambda_j \geq 0, \qquad \forall j \in N, \tag{4.1c}$$

where $N = \{1, \ldots, n\}$ is a set of indices, $\lambda = (\lambda_1, \ldots, \lambda_n)^T$ is the column vector of decision variables, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $a_j \in \mathbb{R}^m$, $\forall j \in N$. Notice that we have changed to λ the symbol used to represent the vector of decision variables, in order to follow the standard notation in the column generation literature.

We assume that the number of variables in the MP is huge when compared to the number of constraints, which makes solving problem (4.1) a difficult task. Furthermore, we assume

the columns a_j are not given explicitly but are implicitly represented as elements of a set $\mathcal{A} \neq \varnothing$, and they can be generated by following a known rule. To solve the MP, we consider only a small subset of columns at first, which leads to an auxiliary problem that is called the *restricted master problem* (RMP):

$$z_{RMP} := \min \quad \sum_{j \in \overline{N}} c_j \lambda_j, \tag{4.2a}$$

$$\text{s.t.} \quad \sum_{j \in \overline{N}} a_j \lambda_j = b, \tag{4.2b}$$

$$\lambda_j \geq 0, \qquad \forall j \in \overline{N}, \tag{4.2c}$$

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

Let $u = (u_1, \ldots, u_m)$ be the vector of dual variables associated to constraints (4.2b) of the RMP. Given an optimal solution of the RMP, represented by the primal-dual pair $(\overline{\lambda}, \overline{u})$, the primal component $\overline{\lambda}$ corresponds to a feasible solution of the MP, as mentioned above. However, the dual component $\overline{u}$ may be infeasible for the MP, because not all the columns are available in the RMP. To check the feasibility of $\overline{u}$ in the MP, we can use the reduced costs $s_j = c_j - \overline{u}^T a_j$, for each $j \in N$. If $s_j < 0$ for some $j \in N$, then the dual solution $\overline{u}_j$ is not feasible and, therefore, $\overline{\lambda}$ cannot be optimal. Otherwise, if $s_j \geq 0$ for all $j \in N$, then $(\overline{\lambda}, \overline{u})$ is also an optimal solution of the MP.

Since we have assumed that the columns a_j are not explicitly available, we should avoid computing the values s_j for all $j \in N$. On the other hand, we are able to obtain the minimum s_j by calling the *oracle*, or *pricing subproblem*,

$$z_{SP}(\overline{u}) := \min\{0, c_j - \overline{u}^T a_j | a_j \in \mathcal{A}\}. \tag{4.3}$$

In some applications, the subproblem (4.3) can be partitioned into several independent subproblems that provide different types of columns. In this case, $z_{SP}(\overline{u})$ corresponds to the minimum reduced cost over all the subproblems.

Using the notation defined above, $z_{SP}(\overline{u}) = 0$ means that no column has a negative reduced cost and, hence, an optimal solution of the MP has been obtained. Otherwise, the column a_j with the minimal reduced cost should be added to the RMP and the problem must be reoptimized. More than one column may be found by solving (4.3), and we can add them all to the RMP. Actually, any column with a negative reduced cost can be added to the RMP. By using the value $z_{SP}(\overline{u})$, we can obtain a lower bound of the optimal value of the MP which is given by $z_{RMP} + \kappa z_{SP}(\overline{u})$, where

$$\kappa \geq \sum_{i \in N} \lambda_i^\star, \tag{4.4}$$

for an optimal solution $\lambda^\star = (\lambda_1^\star, \ldots, \lambda_n^\star)$ of the MP. Indeed, we cannot reduce z_{RMP} by more than κ times $z_{SP}(\overline{u})$. The value of κ is typically known for a master problem formulation. Otherwise, a sufficiently large value may be adopted. In summary, the optimal value of the MP is bounded by

$$z_{RMP} + \kappa z_{SP}(\overline{u}) \leq z^\star \leq z_{RMP}. \tag{4.5}$$

The column generation method consists in exploiting the ideas described above. At each iteration, the optimal solution of the RMP is obtained and the dual component is sent to the oracle. The oracle is in charge of generating one or more columns having negative reduced costs. The method terminates when both bounds in (4.5) are the same or, in other words,

when $z_{SP} = 0$. We refer to the number of RMPs solved as *outer* iterations. The number of iterations to solve a given RMP is called *inner* iterations, and any linear programming algorithm can be applied for this purpose. The standard column generation algorithm is summarized in Algorithm 6.

Algorithm 6: The Standard Column Generation Method.

Input: Initial RMP; parameters κ and $\delta > 0$.
Output: optimal solution λ.

```
1  Set LB = −∞, UB = ∞, gap = ∞;
2  While (gap ≥ δ) do
3      find an optimal solution (λ̄, ū) of the RMP;
4      UB = min(UB, z_RMP);
5      call the oracle with the query point ū;
6      LB = max(LB, z_RMP + κ z_SP(ū));
7      gap = (UB − LB)/(1 + |UB|);
8      if (z̃_SP < 0) then add the new columns to the RMP;
9  end(while)
```

4.2 Variants of the standard column generation method

As discussed in the previous section, the RMP is solved to optimality at each outer iteration of the column generation method. However, this strategy adversely affects the performance of the method, due to the unstable behavior of optimal dual solutions. The situation is even worse when the optimal solutions are extreme points of the dual set. As a result, this typically leads to a large number of outer iterations and slow convergence in the last iterations of the method. In Fig. 4.1, we illustrate the instability of optimal dual solutions throughout four subsequent outer iterations. The figure is split in four parts, each one representing the *dual* feasible set of the corresponding RMPs. An arrow is used to indicate the optimization direction. In part (a) of Fig. 4.1, we represent the initial dual feasible set. The optimal solution in this set is represented by the (yellow) ball in the bottom of the figure. The corresponding optimal value is represented by the dotted (red) line. The optimal dual solution shown in part (a) is sent to the oracle, which generates a new column (represented as a new cut in the dual feasible set). In part (b), we have the new dual feasible set which is obtained after adding this column to the RMP. The illustration in parts (c) and (d) follows the same explanation, but considering the next outer iterations. The purpose of the illustration is to call the attention for the unstable behavior of the optimal dual solutions obtained throughout the iterations. After adding the generated columns (cuts) to the RMP, the dual solution jumps from one side to the other side of the dual feasible region. Apart from the variation, we see that the objective function changes very few, which causes the slow convergence of the method.

Different techniques have been proposed to overcome the instability of the column generation method. They exploit the fact that non-optimal solutions can also be used to generate columns for the RMP. Hence, these techniques rely on interior points of the dual feasible set, which are obtained in a way to avoid the sharp oscillations observed in optimal dual solutions. The *stabilization techniques* were the first variants of the column generation method which used this idea (Marsten et al., 1975; Wentges, 1997; du Merle et al., 1999; Briant et al., 2008; Ben Amor et al., 2009). In general, these techniques use a dual point called stability center and modify the RMP by adding penalization terms and/or artificial variables. The modified RMP is solved to optimality, but the dual solutions are kept relatively close to the stability center and, hence, the variants in subsequent dual solutions are reduced. Good computational

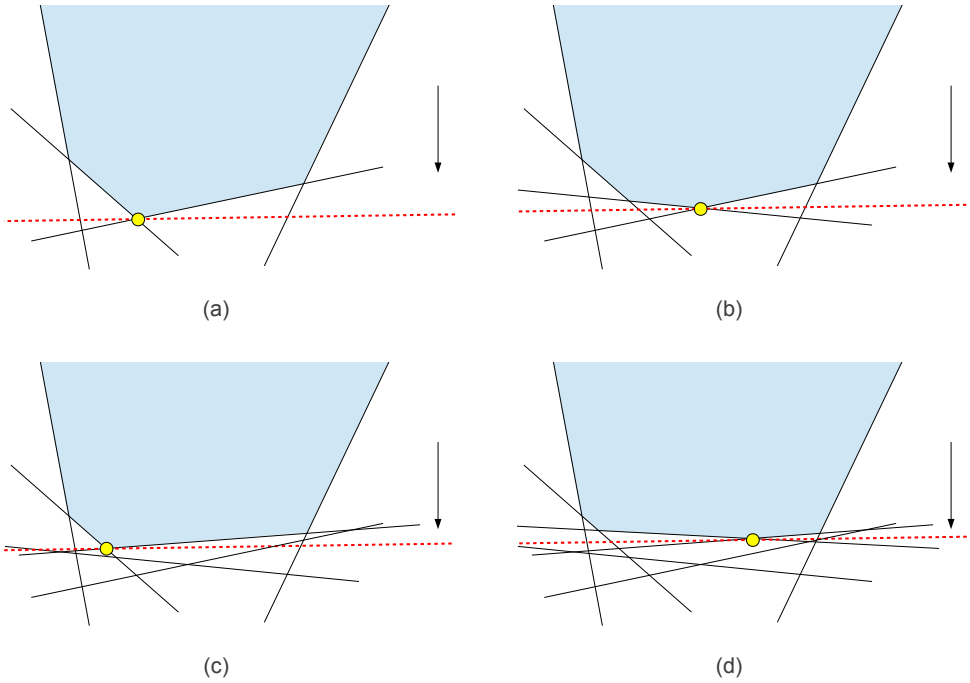

Figure 4.1: Illustration of the unstable behavior of the optimal solutions in four subsequent outer iterations of the standard column generation method.

results are reported for these techniques, although they are dependent of appropriate choices of the stability center and penalization terms. For performance comparisons involving stabilized variants and the standard column generation, see Briant et al. (2008) and Ben Amor et al. (2009). Rousseau et al. (2007) propose an stabilization technique that obtains the dual solutions by solving the RMP repeated times, but using randomly generated objective functions at each time. Then, a set of vertices of the dual space is generated and an interior dual point is given by the convex combination of the points in the set. Promising computational results are presented for the approach, which shows a better overall performance than other stabilization techniques.

Other variants of the column generation method obtain interior points of the dual feasible set without (directly) modifying the RMP. Mitchell and Borchers (1996) and Mitchell (2000) address the solution of two classes of combinatorial optimization problems by a cutting plane method. The strategy obtains the dual points by using the early termination of the interior point algorithm. In those particular applications, the columns are explicitly known in advance, as they come from facet defining valid inequalities. Hence, at each call to the oracle, the columns are generated by full enumeration of the valid inequalities. The analytic center cutting plane method (ACCPM) is another column generation method that uses the interior point algorithm, but it follows a different strategy (Goffin et al., 1992; Atkinson and Vaidya, 1995; Goffin and Vial, 2002). The strategy consists in computing a dual point which is an approximate analytic center of the localization set of to the current RMP. The localization set is given by the intersection of the dual space of the RMP with a half-space given by the best lower bound found for the optimal dual value of the MP. Relying on points in the center of the localization set usually prevents the unstable behavior between consecutive dual points. Moreover, it contributes to the generation of deeper cuts of the dual feasible set. A very important property of this approach is given by its theoretical fully polynomial complexity.

On the other hand, obtaining the analytic center at each iteration of the method may be computationally expensive in practice, specially when many columns are added at a time (Elhedhli and Goffin, 2004). The use of stabilization terms within the ACCPM has been successful, as shown in Babonneau et al. (2007).

The primal-dual column generation method is another variant which uses the interior point algorithm to solve the RMPs (Gondzio and Sarkissian, 1996). The method relies on dual points that are well-centered in the feasible set. However, instead of recurring to the analytic center as in the ACCPM, the primal-dual column generation method uses suboptimal solutions that belong to a safe neighborhood of the central path. The distance of the suboptimal solution to optimality is dynamically adjusted in function of the column generation gap. In the first outer iterations, each RMP is solved with a loose tolerance, and this tolerance is dynamically reduced throughout the iterations as the gap in the column generation approaches zero. Gondzio and Sarkissian (1996) present promising computational results for a class of nonlinear programming problems, whose linearization is solved by column generation. Using a very similar strategy, Martinson and Tind (1999) also report a substantial reduction in the number of outer iterations when compared to other variants of the column generation method. To the best of our knowledge, the primal-dual column generation method has never been applied in the context of integer programming, where column generation formulations are widely employed. Moreover, no theoretical study about the convergence of this method is available in the literature. We attempt to close these two gaps in the study presented in the next section.

4.3 The primal-dual column generation method

The primal-dual column generation method is based on well-centered, non-optimal dual solutions of the RMPs. These dual solutions are provided by the primal-dual interior point algorithm at no extra cost, as this algorithm keeps the iterates close to the central path of the feasible set. As a result, two important features are observed in the primal-dual column generation method. First, the use of non-optimal solutions results in a smaller number of inner iterations, which reduces the CPU time per outer iteration. Second, the well-centered dual points are more stable, which leads to substantial reductions in the number of outer iterations as well as in the total CPU time. In this section, we describe the foundations of the primal-dual column generation method and present a novel theoretical analysis regarding its convergence in a finite number of steps.

Following the notation of Section 4.1, we consider that a given RMP is represented by (4.2) and the optimal primal-dual solution of this problem is given by $(\overline{\lambda}, \overline{u})$. Similarly to the standard approach, the primal-dual column generation method starts with an initial RMP which has enough columns to avoid an unbounded solution. However, in a given outer iteration, the method obtains a suboptimal feasible solution $(\tilde{\lambda}, \tilde{u})$ of the current RMP, which is defined as follows.

Definition 4.3.1. *A primal-dual feasible solution $(\tilde{\lambda}, \tilde{u})$ of the RMP is called suboptimal solution, or ε-optimal solution, if it satisfies $0 \leq (c^T\tilde{\lambda} - b^T\tilde{u}) \leq \varepsilon(1 + |c^T\tilde{\lambda}|)$, for some tolerance $\varepsilon > 0$.*

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

Once the suboptimal solution of the RMP is obtained, the oracle is called with the dual solution $\tilde{u}$ as a query point. Then, it should return either a value $z_{SP}(\tilde{u}) = 0$, if no columns

could be generated from the proposed query point, or a value $z_{SP}(\tilde{u}) < 0$, together with one or more columns to be added to the RMP. Observe that in the second case 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 nonnegative reduced cost.

Consider the value $\kappa > 0$ defined as (4.4). Recall this value depends on the application and it is typically a known parameter. According to Proposition 4.3.2, a lower bound of the optimal value of the MP can still be obtained. It is the classical Lagrangian bound (see e.g. Briant et al., 2008; Ben Amor et al., 2009), but derived from a column generation scheme and using a subpotimal solution.

Lemma 4.3.2. *Let $\tilde{z}_{SP} := z_{SP}(\tilde{u})$ be the value of the oracle corresponding to the suboptimal solution $(\tilde{\lambda}, \tilde{u})$. Then, $\kappa\tilde{z}_{SP} + b^T\tilde{u} \leq z^\star$.*

Proof. Let $\lambda^\star$ be an optimal primal solution of the MP. By using (4.1b) and $\tilde{z}_{SP} \leq 0$, we have that

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

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

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

$$\gamma\mu \leq (c_j - u^T a_j)\lambda_j \leq (1/\gamma)\mu, \quad \forall j \in \overline{N}, \tag{4.6}$$

for some $\gamma \in (0.1, 1]$, where $\mu = (1/|\overline{N}|)(c^T - u^T A)\lambda$. By imposing (4.6), we require the point to not be too close to the boundary of the primal-dual feasible set and, hence, the oscillation of the dual solutions will be relatively small. Notice that (4.6) is a natural way of stabilizing the dual solutions, if a primal-dual interior point method is used to solve the RMP. Indeed, the requirement in (4.6) is the same as that used in the symmetric neighborhood of the central path (see Chapter 2, Section 2.3).

Fig. 4.2 illustrates the behavior of well-centered, suboptimal solutions. In part (a) we have the dual feasible set of a given RMP. The central path of the dual feasible set is given by the curve in the middle of the feasible region (the reader should bear in mind it is just an illustration with the purpose of clarifying the theoretical background). The dashed lines close to the central path are the bounds of the safe neighborhood of the central path. As defined in (4.6), the dual solutions must belong to this neighborhood. The suboptimal solution obtained by the interior point method is represented by a (green) triangle. By using this solution, the oracle generates a new column to be added to the RMP. The cut corresponding to this column is represent by the (blue) dashed line in part (b). Notice it is a much deeper cut than the one that would be generated by using the optimal dual solution (see Fig. 4.1). It results in cutting-off a larger portion of the dual feasible region. After adding the generated column to the RMP, we obtain the dual feasible set given in part (c).

In order to obtain a suboptimal solution $(\tilde{\lambda}, \tilde{u})$, we need to set the tolerance ε which defines the distance of the suboptimal solution to optimality. We propose to adjust this parameter dynamically, based on the following observations. The tolerance ε can be loose at the beginning of the column generation process, as a very rough approximation of the MP is known at this time. Throughout the outer iterations, this tolerance should be gradually reduced so that it becomes tight when the gap is small. Let *gap* denote the column generation

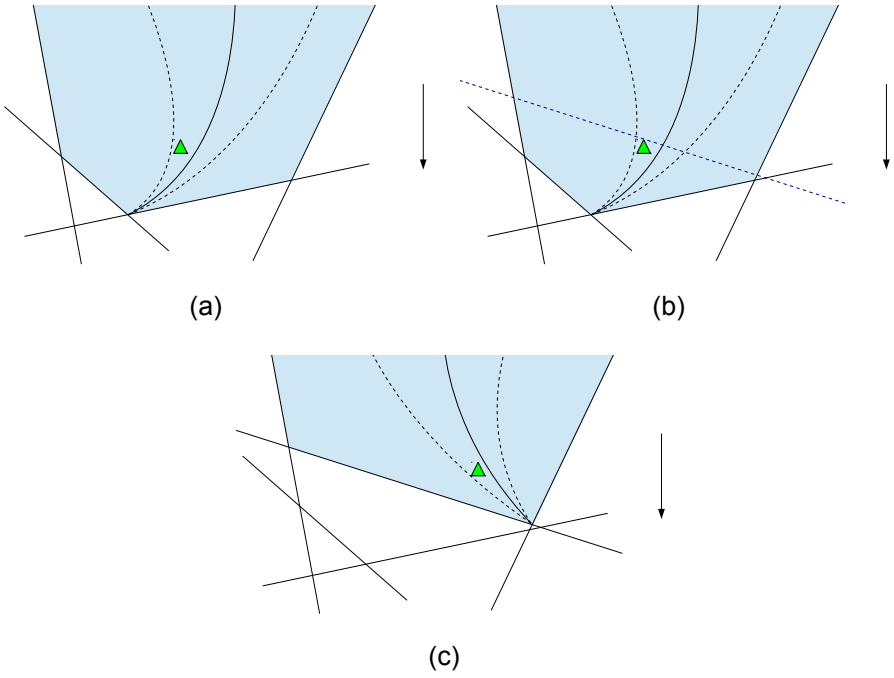

Figure 4.2: Illustration of the behavior of the well-centered, suboptimal solutions in two subsequent outer iterations of the primal-dual column generation method.

relative gap, which is given by

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

where $\tilde{z}_{SP} := z_{SP}(\tilde{u})$, as defined in Proposition 4.3.2. We propose to set ε in function of the relative gap, as follows. In a given outer iteration k, we recompute the relative gap so that the tolerance ε^k is updated as

$$\varepsilon^k := \min\{\varepsilon_{\max},\ gap^{k-1}/D\}, \tag{4.7}$$

where $D > 1$ is the *degree of optimality* that relates the tolerance ε^k to the relative gap at iteration $k-1$. Here, we consider that D is a fixed parameter. Notice that an upper bound $\varepsilon_{\max}$ is used so that the suboptimal solution is not too far from the optimum (e.g., $\varepsilon_{\max} = 1.0$).

It is important to emphasize that unlike in the standard approach, $\tilde{z}_{SP} = 0$ does not suffice to terminate the column generation process. Proposition 4.3.3 shows that the gap is still reduced in this case, and the progress of the algorithm is guaranteed.

Lemma 4.3.3. *Let $(\tilde{\lambda}, \tilde{u})$ be the suboptimal solution of the RMP, found at iteration k with tolerance $\varepsilon^k > 0$. If $\tilde{z}_{SP} = 0$, then the new relative gap is strictly smaller than the previous one, i.e., $gap^k < gap^{k-1}$.*

Proof. We have that $\tilde{z}_{RMP} = c^T\tilde{\lambda}$ is an upper bound of the optimal solution of the MP. Also, from Proposition 4.3.2 we obtain the lower bound $b^T\tilde{u}$, since $\tilde{z}_{SP} = 0$. Hence, the gap in the current iteration is given by

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

Additionally, from Definition 4.3.1, we know that $gap^k \leq \varepsilon^k$. Finally, from (4.7) and since $D > 1$, we have that $\varepsilon^k < gap^{k-1}$ which completes the proof. □

Algorithm 7 summarizes the above discussion. Notice that the primal-dual column generation method has a simple algorithmic description, similar to the standard approach. Thus, it can be implemented in the same level of difficulty if a primal-dual interior point solver is readily available. Notice that κ is known in advance and problem dependent. Also, the upper bound of the RMP, $\tilde{z}_{RMP}$, may slightly increase from one iteration to another due to the use of suboptimal solutions and, hence, we store the best value found so far in UB (step 5).

Algorithm 7: The Primal-Dual Column Generation Method.

Input: Initial RMP; parameters κ, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
Output: optimal solution λ.

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

Since the primal-dual column generation method relies on suboptimal solutions of each RMP, it is important to ensure that it is a valid column generation procedure, *i.e.*, a finite iterative process that delivers an optimal solution of the MP. Even though the optimality tolerance ε decreases geometrically in the algorithm, there is a special case in which the subproblem value $\tilde{z}_{SP}$ may be zero. This would cause the method to stall and, hence, this situations requires an appropriate treatment. Fortunately, by using Proposition 4.3.3 we can guarantee the method still converges successfully. The proof of convergence is given in Proposition 4.3.4.

Theorem 4.3.4. *Let $z^\star$ be the optimal value of the MP. Given $\delta > 0$, the primal-dual column generation method converges in a finite number of steps to a primal feasible solution $\hat{\lambda}$ of the MP with objective value $\tilde{z}$ that satisfies*

$$(\tilde{z} - z^\star) < \delta(1 + |\tilde{z}|). \tag{4.8}$$

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

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

2. $\tilde{z}_{SP} = 0$ and no columns have been generated. If additionally we have $\varepsilon^k < \delta$, then from Proposition 4.3.3 the gap in the current iteration satisfies $gap^k < \delta$, and the algorithm terminates with the suboptimal solution $(\tilde{\lambda}, \tilde{u})$. Otherwise, we also know from Proposition 4.3.3 that the gap is still reduced, and although the RMP in the next iteration will be the same, it will be solved to a tolerance $\varepsilon^{k+1} < \varepsilon^k$. Moreover, the gap is reduced by a factor of $1/D$ and, hence, after a finite number of iterations we obtain a gap less than δ.

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

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

Since $\kappa \tilde{z}_{SP} + b^T \tilde{u} \leq z^\star$, the inequality (4.8) is satisfied with $\tilde{z} = \tilde{z}_{RMP}$. The primal solution $\tilde{\lambda}$ leads to a primal feasible solution of the MP, given by $\hat{\lambda}_j = \tilde{\lambda}_j$, $\forall j \in \overline{N}$, and $\hat{\lambda}_j = 0$, otherwise. If $gap^k \geq \delta$, a new iteration is carried out and we have one of the above situations again. □

Having presented a proof of convergence for the primal-dual column generation method, it is important to give some remarks about its implementation. First of all, it is important to rely on a state-of-the-art interior point solver. Notice that a primal-dual interior point method is well-suited to this purpose. In fact, (standard) simplex type methods cannot straightforwardly provide suboptimal solutions which are well-centered in the dual space. Instead, the primal and dual solutions are always at the boundaries of their corresponding feasible sets. Besides, there is no control on the infeasibilities of the solutions before optimality is reached in a simplex method. In our implementation, each RMP is solved by the interior point solver HOPDM (Gondzio, 1995). It keeps the iterates inside a neighborhood of the central path, which has the form (4.6). Besides, the solver makes use of multiple centrality correctors in case a more strict centrality is required by the user (Gondzio, 1996; Colombo and Gondzio, 2008).

An efficient warmstarting technique is another essential feature in the primal-dual column generation method. Throughout the column generation iterations, closely-related problems are solved, as the RMP in a given iteration differs from the RMP of the previous iteration by merely a few columns. Hence, this similarity should be exploited in order to reduce the computational effort of solving a sequence of problems. In our implementation of PDCGM, we rely on the warmstarting techniques available in the solver HOPDM (see Gondzio (1998); Gondzio and Grothey (2003, 2008)). The main idea of these methods consists of storing a close-to-optimality and well-centered iterate when solving a given RMP. After a modification is carried out on the RMP, the stored point is used as a good initial point to start from. See Chapter 2, Section 2.3.1, for further details on warmstarting strategies for the primal-dual interior point method.

4.4 Computational results

In this section, we present a computational study of the primal-dual column generation method using three classical problems from the literature. They are the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST). For each problem, we have implemented three different column generation variants. The descriptions of each variant are the following:

- Standard column generation method (SCGM): each RMP is solved to optimality by a simplex-type method to obtain an (extremal) optimal solution. We have used the best available linear programming solver in the package IBM ILOG CPLEX v.12.1 (2010) to obtain such a solution. Preliminary tests with this package using the default settings for each solver have shown 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 suboptimal solutions of each RMP are obtained by using the interior point solver HOPDM (Gondzio, 1995), which is able to efficiently provide well-centered dual points.

- Analytic center cutting plane method (ACCPM): the dual point at each iteration is an approximate analytic center of the localization set associated to the current RMP. The implementation was carried out on top of the open-source solver OBOE (COIN-OR, 2010), a state-of-the-art implementation of the analytic center cutting plane strategy with additional stabilization terms (Babonneau et al., 2007).

For each problem and for every aforementioned column generation variant, the subproblems are solved with the same source-code. 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 a better performance of the method on average. We have used different initial dual points for each problem, as will be specified later. To run the tests we have used a computer with processor Intel Core 2 Duo 2.26 Ghz, 4 GB RAM, and Linux operating system. For each of the methods, we stop the column generation procedure when the relative gap becomes smaller than the default accuracy $\delta = 10^{-6}$.

The purpose of comparing the PDCGM against the SCGM is to give an idea of how much it can be gained in overall performance in relation to the standard approach (*i.e.*, extreme dual solutions without any stabilization). Undoubtedly, it would be interesting to consider stabilized versions of the standard column generation in the computational study presented here. However, the lack of publicly available codes of stabilized versions discouraged us to include them to this computational study. For the interested reader, available comparisons between standard and stabilized column generation methods are available in the literature for the same problems we consider here (Rousseau et al., 2007; Briant et al., 2008; Ben Amor et al., 2009). The ACCPM was included in our experiments for being a strategy that also relies on an interior point method (although essentially different). After extensive testing we chose what seems to be the best possible starting point/parameters setting for the OBOE solver. To clarify the main differences regarding the three variants, Fig. 4.3 illustrates the dual solutions used by the each of them. Two dual feasible sets are represented in the figure. In each of them, the (yellow) ball represents the dual point used by the SCGM, *i.e.*, an (extreme) optimal solution. The (green) triangle represents the well-centered, suboptimal solution which is used in the PDCGM. A (red) square is used to represent the approximate analytic center used in the ACCPM. From (a), we have the dual feasible region of an RMP which has a unique optimal dual solution. The dual feasible region from (b) illustrates the case with multiple optimal dual solutions (primal degeneracy). The extreme optimal dual solution obtained by the SCGM can be any of the (yellow) balls in the figure, which are vertices of the region. It is worth mentioning that we have additionally tested the performance of using the interior point algorithm in a column generation method in which each RMP is solved to optimality. The results were inferior to the ones obtained by PDCGM, which shows that an appropriate use of an interior point method is essential for its success in the column generation context.

In the problems addressed in this study, the master problem formulations are obtained by applying the Dantzig-Wolfe decomposition (DWD) to standard integer programming formulations (see Appendix A). In each application, the decomposition leads to an integer MP and also an integer (pricing) subproblem. Here, we relax the integrality of the variables in the integer MP and then solve it by column generation, which gives a lower bound of the optimal value of the original formulation. To obtain an integer solution, it would be necessary to combine the column generation with a branch-and-bound search, which is known as a branch-

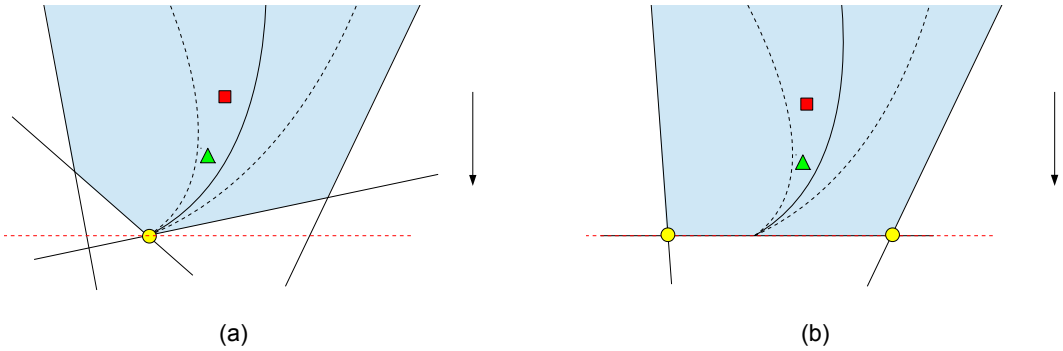

Figure 4.3: Illustration of the dual solutions used by each column generation variant.

and-price method (Barnhart et al., 1998; Lübbecke and Desrosiers, 2005). This combination is out of the scope of this computational study, as we are concerned with the behavior of the column generation variations. The use of the primal-dual column generation method within a branch-and-price methodology is addressed in Chapter 5.

4.4.1 Cutting stock problem

The one-dimensional CSP consists in determining the smallest number of stock rolls of fixed width W that have to be cut in order to satisfy the demands of m different pieces of smaller widths (Gilmore and Gomory, 1961, 1963). The coefficient matrix of its standard integer programming formulation has a special structure which is well-suited to the application of the DWD (see Appendix A for a complete description of the formulation and its decomposition). The oracle associated to the decomposition is given by a set of n subproblems, where n is an upper bound for the total number of stock rolls. We assume the rolls have all the same width W and hence the subproblems are identical. This feature leads to an aggregated master problem with $m+1$ constraints and the oracle reduces to only one subproblem, which corresponds to an integer knapsack problem. If the k-best solutions of the knapsack problem are available, for a given $k > 0$, then up to k columns can be generated at each call to the oracle. It usually improves the performance of the column generation procedure, since more information is gathered at each iteration.

To analyze the performance of the column generation variants addressed in this computational study, we have initially selected 262 instances from the literature in one-dimensional CSP (`http://www.math.tu-dresden.de/~capad/`). We have classified these instances into two classes according to m, the number of pieces. Class S (small size instances) includes 178 instances with dimensions ranging from 15 to 199 pieces, while class M (medium size instances) has 84 instances ranging from 200 to 555 pieces. In the computational experiments, the initial RMP consists of columns that are generated from 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 proposed by Leão (2009), which implementation was provided by the author. The method is able to provide the k-best solutions of the knapsack problem.

For each class of instances, we have tested the column generation variations using four different values of k (1, 10, 50 and 100), the maximum number of columns added to the RMP at each outer iteration. Table 4.1 summarizes the obtained results. For each variation we have three columns: the average number of outer iterations (Iter), the average CPU time spent in

the oracle (Oracle) and the average CPU time required for the column generation procedure (Total). For each k, the row *All* presents the average results over the 262 instances using the same k. The last three columns in Table 4.1 show the average total time per iteration (in seconds) and the oracle times are shown in parentheses. The time spent by the SCGM in solving the RMPs is very small in relation to the time required to solve the subproblems, regardless the size of the instances. It happens because the simplex method available in the CPLEX solver is very efficient on solving/reoptimizing these linear programming problems. For the PDCGM and the ACCPM, the proportion of the total CPU time required to solve the RMP and the oracle varies according to the size of the instances.

		SCGM			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
k	Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCGM	PDCGM	ACCPM
1	S	571.8	2.0	2.6	368.9	1.9	4.8	466.2	3.0	10.5	$< 0.01 (< 0.01)$	$0.01 (< 0.01)$	0.02(0.01)
	M	881.3	153.7	155.8	591.4	35.5	44.5	734.0	143.9	182.4	0.18(0.17)	0.08(0.06)	0.25(0.20)
	All	671.0	50.6	51.7	440.3	12.7	17.5	552.1	48.2	65.6	0.08(0.07)	0.04(0.03)	0.12(0.09)
10	S	149.7	0.9	1.2	102.2	0.8	2.1	253.1	2.5	26.1	0.01(0.01)	0.02(0.01)	0.10(0.01)
	M	251.4	75.8	77.0	158.4	15.1	18.3	368.3	82.6	148.7	0.31(0.30)	0.12(0.10)	0.40(0.22)
	All	182.3	24.9	25.5	120.2	5.4	7.3	290.0	28.2	65.4	0.14(0.14)	0.06(0.04)	0.23(0.10)
50	S	70.9	1.8	2.1	63.2	2.0	3.8	276.8	10.7	106.3	0.03(0.03)	0.06(0.03)	0.38(0.04)
	M	133.7	56.6	58.2	97.1	18.8	23.1	400.2	45.5	277.6	0.44(0.42)	0.24(0.19)	0.69(0.11)
	All	91.0	19.4	20.1	74.1	7.4	10.0	316.4	21.9	161.2	0.22(0.21)	0.13(0.10)	0.51(0.07)
100	S	53.7	3.8	4.2	53.9	4.6	7.3	308.4	31.2	221.8	0.08(0.07)	0.14(0.09)	0.72(0.10)
	M	101.0	66.3	67.8	82.3	25.4	31.5	449.4	96.4	525.2	0.67(0.66)	0.38(0.31)	1.17(0.21)
	All	68.8	23.9	24.6	63.0	11.3	15.1	353.6	52.1	319.1	0.36(0.35)	0.24(0.18)	0.90(0.15)

Table 4.1: Average results on the 262 CSP instances using different values for k (maximum number of columns added to the RMP at a time).

Regardless the number of columns added at each outer iteration (k), the PDCGM has the smallest number of iterations on average. For instances in class M, the PDCGM is on average more efficient than the SCGM and the ACCPM in terms of CPU time as well. For example, if we consider $k = 100$, the PDCGM is 2.2 times faster than the SCGM and 16.7 times faster than the ACCPM in the medium size instances. We observe similar results by considering all the instances. Indeed, the PDCGM is 1.6 times faster than the SCGM and 21.1 times faster than the ACCPM on average, also for $k = 100$. Only in the class of small instances (S), we observe that the SCGM is more efficient than the PDCGM and the ACCPM regarding the average CPU time.

The results in Table 4.1 indicate that PDCGM with $k = 10$ is the variant with the best overall performance regarding CPU time. On average, it is 2.8 and 8.9 times faster than the best scenarios for the SCGM ($k = 50$) and the ACCPM ($k = 10$), respectively. The behavior 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 behavior 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 center can be very expensive in this case. Another important observation is that the average CPU time per iteration spent in the oracle is smaller for the PDCGM and the ACCPM than for the SCGM, which is a consequence of using well-centered, stable dual solutions. Furthermore, the PDCGM seems to be able to solve the RMPs more efficiently than the ACCPM and, hence, this variant achieved better total average CPU times per iteration.

We further compare the performance of the three column generation variants by reporting the results on 14 additional instances, in which m, the number of items, varies from 615 to 1005. Thus, we have large restricted master problems in this experiment. Table 4.2 shows

the results of each column variant using $k = 100$. In all cases, the PDCGM is faster and requires less 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.

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

Table 4.2: Results on 14 large CSP instances (with $k = 100$).

Finally, we extend our computational study by presenting the results obtained with 160 instances from the so-called triplet and uniform sets proposed in Falkenauer (1996). Table 4.3 shows the results in which 100 columns are added at each outer iteration. We have grouped the instances in three classes: $S(u,t)$, 80 small instances with m varying from 60 to 120; $M(u,t)$, 120 medium instances with m between 249 and 501; and $L(u,t)$, 20 large instances with $m = 1000$. For the last class of instances, the ACCPM was not able to solve all the instances so we have omitted the corresponding results in the last row. The columns in Table 4.3 have the same meaning as in Table 4.1. For these instances, the SCGM performs better than the PDCGM and the ACCPM in classes $S(u,t)$ and $M(u,t)$. However for large instances ($L(u,t)$), PDCGM outperforms the SCGM in both CPU time and number of outer iterations.

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

Table 4.3: Average results on 220 CSP instances from triplet and uniform problem sets (using $k = 100$).

4.4.2 Vehicle routing problem with time windows

Consider a set of vehicles available in a depot to serve n customers with known demands. A vehicle can serve more than one customer in a route, as long as its maximum capacity is not exceeded. Each customer must be served once within a given time window. Besides, a service time is assigned for each customer. Late arrivals are not allowed and if a vehicle arrives earlier to a customer it must wait until the time window is open. We assume all the vehicles are identical and are initially at the same depot, and every route must start and finish at this depot. The objective is to design a set of minimum cost routes in order to serve all the customers. The column generation method has been successfully used within integer programming methodologies in the solution of the VRPTW (Desrochers et al., 1992; Kallehauge et al., 2005). In the master problem formulation adopted here, the subproblem

is given by a non-elementary shortest path problem with resource constraints. Similarly to CSP, an aggregated master problem is used, as we consider identical vehicles. The restricted master problems have the set covering structure and its number of rows is equal to $n + 1$. See Appendix A for further description about the problem formulation and the associated decomposition.

We have selected 87 VRPTW instances from the literature (`http://www2.imm.dtu.dk/~jla/solomon.html`), which were originally proposed by Solomon (1987). We have divided them in three classes: S (small size instances, $n = 25$), M (medium size instances, $n = 50$) and L (large size instances, $n = 100$) classes. Each class has 29 instances. The initial columns of the RMP have been generated by n single-customer routes which correspond to assigning one vehicle per customer. In the ACCPM approach, we have considered the initial guess $u^0 = 100.0e$ which after testing various settings has proven to be the choice which gives the best overall results for this problem. The subproblem is solved by our own implementation of the bounded bidirectional dynamic programming algorithm using state-space relaxation and identification of unreachable nodes (Feillet et al., 2004; Righini and Salani, 2008). The implemented solver is able to provide the k-best solutions of the subproblem.

Table 4.4 presents the performance of the three column generation variants when solving the master problem formulation of the VRPTW. Column k denotes the maximum number of columns added to the RMP at each outer iteration. For each column generation variant we have: the number of outer iterations (Iter),the average CPU time to solve the subproblems (Oracle) and the average total CPU time required for the column generation (Total). For each value of k, the last row (All) shows the average results over the 87 instances using the same value of k. In the last three columns, the table shows the average Total and Oracle times per iteration.

		SCGM			PDCGM			ACCPM			Total/Iter (Oracle/Iter)		
			Time (in s)			Time (in s)			Time (in s)				
k	Class	Iter	Oracle	Total	Iter	Oracle	Total	Iter	Oracle	Total	SCGM	PDCGM	ACCPM
1	S	99.9	0.8	0.8	48.7	0.4	0.5	106.6	0.4	0.6	0.01(0.01)	0.01(0.01)	0.01(< 0.01)
	M	279.6	20.0	20.1	101.3	6.1	6.3	162.4	7.5	7.8	0.07(0.07)	0.06(0.06)	0.05(0.05)
	L	797.8	469.7	470.4	213.7	127.8	128.6	292.2	163.4	164.8	0.59(0.59)	0.60(0.60)	0.56(0.56)
	All	392.4	163.5	163.8	121.3	44.8	45.1	187.1	57.1	57.8	0.42(0.42)	0.37(0.37)	0.31(0.31)
10	S	26.2	0.3	0.3	22.3	0.2	0.2	93.6	0.4	0.5	0.01(0.01)	0.01(0.01)	0.01(< 0.01)
	M	66.7	6.1	6.2	37.7	2.4	2.6	122.0	5.4	5.7	0.09(0.09)	0.07(0.06)	0.05(0.04)
	L	188.2	113.5	114.1	72.6	41.0	41.6	170.6	90.9	92.1	0.61(0.60)	0.57(0.56)	0.54(0.53)
	All	93.7	40.0	40.2	44.2	14.5	14.8	128.7	32.2	32.8	0.43(0.43)	0.33(0.33)	0.25(0.25)
50	S	14.4	0.2	0.2	18.1	0.1	0.2	92.2	0.4	0.5	0.01(0.01)	0.01(0.01)	0.01(< 0.01)
	M	33.1	3.5	3.5	26.4	1.6	1.8	120.0	5.3	5.7	0.11(0.11)	0.07(0.06)	0.05(0.04)
	L	88.0	54.9	55.5	48.6	26.0	27.0	165.2	85.8	87.8	0.63(0.62)	0.56(0.53)	0.53(0.52)
	All	45.2	19.5	19.7	31.0	9.3	9.7	125.8	30.5	31.3	0.44(0.43)	0.31(0.30)	0.25(0.24)
100	S	12.2	0.2	0.2	16.7	0.1	0.2	92.3	0.4	0.6	0.02(0.02)	0.01(0.01)	0.01(< 0.01)
	M	26.0	2.9	3.0	23.2	1.4	1.7	119.7	5.4	5.8	0.12(0.11)	0.07(0.06)	0.05(0.05)
	L	65.4	41.7	42.4	37.9	20.3	21.5	166.0	84.5	87.5	0.65(0.64)	0.57(0.54)	0.53(0.51)
	All	34.5	14.9	15.2	25.9	7.3	7.8	126.0	30.1	31.3	0.44(0.43)	0.3(0.28)	0.25(0.24)
200	S	9.8	0.1	0.2	16.1	0.1	0.3	92.4	0.4	0.6	0.02(0.01)	0.02(0.01)	0.01(< 0.01)
	M	20.8	2.3	2.4	21.1	1.2	1.7	120.9	5.3	6.0	0.12(0.11)	0.08(0.06)	0.05(0.04)
	L	50.1	33.1	33.9	32.4	16.9	18.7	167.4	82.1	89.2	0.68(0.66)	0.58(0.52)	0.53(0.49)
	All	26.9	11.9	12.1	23.2	6.1	6.9	126.9	29.3	31.9	0.45(0.44)	0.30(0.26)	0.25(0.23)
300	S	9.4	0.1	0.1	15.8	0.1	0.3	93.3	0.4	0.6	0.01(0.01)	0.02(0.01)	0.01(< 0.01)
	M	18.0	2.1	2.2	20.4	1.2	1.8	121.1	5.2	6.1	0.12(0.12)	0.09(0.06)	0.05(0.04)
	L	42.6	28.7	29.7	31.5	16.2	18.8	168.7	79.4	89.9	0.70(0.67)	0.60(0.51)	0.53(0.47)
	All	23.3	10.3	10.7	22.6	5.8	7.0	127.7	28.3	32.2	0.46(0.44)	0.31(0.26)	0.25(0.22)

Table 4.4: Average results on 87 VRPTW instances adding at most k columns at a time.

In all the classes, the PDCGM shows the best average performance regarding the number of iterations and total CPU time. When the size of the instances increases, the difference between the SCGM and the other two methods increases as well, with the SCGM being the

one which shows the worst overall performance. For $k = 1$, the PDCGM is on average 3.7 and 1.3 times faster than the SCGM and the ACCPM, respectively. For larger values of k, the SCGM and the PDCGM have a similar overall performance in the class of small instances (S). But, if we take into account classes M and L, the PDCGM seems to be consistently more efficient than the other two approaches in both, number of outer iterations and total CPU time, for any k. The same conclusion is obtained considering all the 87 instances. For all the strategies and values of k, the PDCGM with $k = 200$ is the most efficient setting on average. This variant is 1.6 and 4.5 times faster than the best results obtained with the SCGM ($k = 300$) and the ACCPM ($k = 100$), respectively.

The results in Table 4.4 indicate that the well-centered dual points provided by the PDCGM and the ACCPM lead to smaller average oracle CPU times per iteration when compared to the SCGM. Differently from what was observed on the CSP results, the CPU time required for solving the RMPs is very small not only for the SCGM, but also for the PDCGM and the ACCPM. The ACCPM achieved now the best results regarding the total CPU times per iteration, as it can efficiently solve the RMPs in this case, even if up to 300 columns are added at each call to the oracle. The only drawback of this strategy was the (relatively) large number of outer iterations.

We have tested the three column generation methods in more challenging instances, which were proposed by Homberger and Gehring (2005). This set of instances includes problems with 200, 400 and 600 customers, so larger RMPs are obtained with them. Table 4.5 shows the results of this additional experiment, in which up to 300 columns are added to the RMP at each outer iteration. Column n denotes the number of customers per instance while the remaining columns have the same meaning as in Table 4.4. For all instances, the PDCGM requires less CPU time and fewer iterations when compared with the SCGM and the ACCPM. For the most difficult instance, namely RC1_6_1, the PDCGM is 2.1 and 6.4 times faster than the SCGM and the ACCPM, respectively. In terms of time per iteration, the three strategies behave similarly, in general.

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

Table 4.5: Results on 9 large VRPTW instances adding 300 columns at a time.

4.4.3 Capacitated Lot-Sizing Problem with Setup Times

Consider m items which must be processed by a single machine in n time periods. The objective is to minimize the total cost of producing, holding and setting up the machine in order to satisfy the demands of each item at each time period. Processing and setup times are associated to the manufacturing of each item and the machine has a limited capacity. This problem is known as the capacitated lot sizing problem with setup times (CLSPST) (see Trigeiro et al., 1989; Jans and Degraeve, 2004). A detailed description of formulation and the decomposition regarding this problem can be found in Appendix A. Each subproblem is a single-item lot sizing problem with modified production and setup costs, and without

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

* 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 $\delta = 10^{-5}$ for solving those instances.

Table 4.6: Average results on 751 CLSPST instances.

capacity constraints. Hence, it can be solved by the Wagner-Whitin algorithm (Wagner and Whitin, 1958). Unlike the other two applications, the oracle in the CLSPST consists of m different subproblems and the master problem has a disaggregated formulation. As a result, the master problem has $n + m$ rows and up to m columns may be generated at each outer iteration, one from each subproblem.

We have selected 751 CLSPST instances proposed by Trigeiro et al. (1989) to test the aforementioned column generation variants. 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 set to 1 in the convexity constraints. In the ACCPM approach, after several settings, we have chosen $u^0 = 10.0e$ as the initial dual point. The subproblems are solved using our own implementation of the Wagner-Whitin algorithm.

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 item 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, so all instances remain feasible. In Table 4.6, we show a summary of our findings using the modified instances. We have grouped the instances into 7 different classes. Small size instances are included in classes E, F and W, while classes $G, X1, X2$ and $X3$ contain medium size instances. For each class and strategy we present: the number of outer iterations (Iter), the average CPU time to solve the subproblems (Oracle) and the average total CPU time required for the column generation (Total). The last row (All) shows the average results considering the 751 modified instances. Additionally to our usual notation, we have included the number of instances per class (Inst).

From Table 4.6, we observe that the column generation variants have different performances for the small instances (classes E, F and W). On average, each variant 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. If we look at the oracle times, we will observe that for small instances the ACCPM and the PDCGM outperform the SCGM due to the reduction in the number of outer iterations. Now, if we observe the performance in classes containing large instances (G, $X1$, $X2$ and $X3$), the PDCGM outperforms the other two strategies, on average. Furthermore, for the 751 instances (All), the PDCGM has the smallest average number of outer iterations and total CPU time.

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

Table 4.7: Average results on 44 CLSPST large instances.

In addition to the previous experiment, we have run the methods on a set with larger instances. We have select 11 challenging instances from sets G, $X2$ and $X3$, namely $G30$, $G53$, $G57$, $X21117A$, $X21117B$, $X21118A$, $X21118B$, $X31117A$, $X31117B$, $X31118A$, $X31118B$. These instances have been replicated 5, 10, 15 and 20 times following the same procedure described above. The summary of our findings are presented in Table 4.7, 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. Considering the 44 instances (11 instances and 4 values for r), the PDCGM is 2.8 and 2.6 times faster than the SCGM and the ACCPM, respectively. If we consider the average CPU time per iteration, then the PDCGM is the most efficient among the variants, while the SCGM and the ACCPM have very similar times per iteration.

4.4.4 Additional computational results in the literature

In Briant et al. (2008), the authors present a comprehensive computational study comparing the standard column generation against the bundle method, a stabilized cutting plane method which uses quadratic stabilization terms. A set of five different applications were used in the computational experiments, including the CSP and the CLSPST (MILS problem in their paper). Regarding the CSP, the results indicate that using the bundle stabilization may slightly reduce the number of iterations at the cost of worsening CPU times. In the results obtained for the CLSPST, the authors report that the bundle method behaves poorly in terms of both, CPU times and number of iterations, when compared with the standard column generation.

Rousseau et al. (2007) propose an interior point column generation technique in which the dual solutions are convex combinations of extreme dual points of the RMP. To analyze the computational performance of their approach, they have used the set of VRPTW Solomon's instances with $n = 100$. Only the results of 22 out of 29 instances were presented by the authors. Their comparison involved the implementations of the standard column generation as well as a stabilized version called BoxPen technique (du Merle et al., 1999). Since a different subproblem solver has been used in their computational experiments, it would not be appropriate to make a straightforward comparison of the figures presented in their tables with those presented in Table 4.4. Hence, we consider the gain obtained by each approach in relation to the standard column generation. According to their results, a well-tuned implementation of the BoxPen stabilization reduces the number of outer iterations by 16%, on average, while the total difference regarding CPU times is negligible. The interior point column generation technique proposed by the authors showed a better performance than the BoxPen stabilization, being 1.38 times faster than the standard column generation technique. For the same set of instances, the PDCGM is around 2 times faster than the SCGM on average ($k = 100$).

4.5 Concluding remarks

In this chapter, we have presented new developments in theory and applications of the primal-dual column generation method (PDCGM). The method relies on the primal-dual interior point method to obtain non-optimal, well-centered solutions of the RMPs. Theoretical support is given to show that the PDCGM converges to an optimum of the master problem, even though non-optimal dual solutions are used. Also, computational experiments show that the method is competitive when compared against the standard column generation method (SCGM) and the analytic center cutting plane method (ACCPM). The experiments were based on linear relaxations of integer master problems formulations of three widely studied integer programming problems, namely the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW) and the capacitated lot sizing problem with setup times (CLSPST). Different types of master problem formulations are used on these applications: an aggregated master problem in the CSP, an aggregated master problem with a set covering structure in the VRPTW, and a disaggregated master problem in the CLSPST. Additionally, we have tested the addition of different numbers of columns at each outer iteration, which typically affects the behavior of the methods.

By analyzing the computational results, we conclude that the PDCGM achieves the best overall performance when compared to the SCGM and the ACCPM. 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 against the SCGM gives an idea of how much can be gained by using non-optimal, well-centered 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-centered interior point solutions is a very attractive feature of this column generation approach.

Several avenues are available for further studies involving the primal-dual column generation technique. One of them is to compare the performance of the PDCGM with advanced column generation variants such as generalized bundle methods and the volume algorithm (Frangioni, 2002; Barahona and Anbil, 2000). Furthermore, since the PDCGM relies on an interior point method, the investigation of new effective warmstarting strategies applicable in this context is essential for the success of the framework. In the next chapter, we address the use of the primal-dual column generation method within a branch-price-and-cut framework.

Chapter 5

Using the primal-dual interior point algorithm within the branch-price-and-cut method

* A compact version of this chapter has been accepted for publication in the journal *Computers & Operations Research* (see Munari and Gondzio, 2013).

The branch-price-and-cut method has been widely used for solving integer programming models in which a special structure can be identified in the coefficient matrix. This structure is exploited by a reformulation technique, e.g. the Dantzig-Wolfe decomposition, that usually leads to a formulation with a stronger linear relaxation when compared with the linear relaxation of the original model. By using this stronger formulation within a branch-and-bound tree, we obtain the branch-and-price method. Since the reformulation may have a huge number of variables, the column generation algorithm is used to solve the linear relaxation at each node of the tree. For this reason, the branch-and-price method is also known as integer programming column generation. In some cases, valid inequalities should also be added to the reformulated model to get even better bounds from the linear relaxations with the aim of improving the branch-and-bound search, which leads to the branch-price-and-cut method. For comprehensive surveys on these methods, see Barnhart et al. (1998); Lübbecke and Desrosiers (2005); Vanderbeck and Wolsey (2010).

In the vast majority of branch-price-and-cut implementations presented in the literature, a simplex-type method is used to solve the linear programming problems at each node. Hence, the generation of columns and valid inequalities are typically based on optimal solutions which are extreme points of the problem feasible set. As it was discussed in Chapter 4, optimal dual solutions adversely affect the performance of the column generation method, as they oscillate sharply in subsequent iterations of the method. To overcome this unstable behavior, more efficient variants of the column generation method use non-optimal dual solutions. The generation of valid inequalities by using non-optimal primal solutions has shown to be more effective as well, since deeper cuts are obtained and a smaller number of them are usually needed.

Although very successful in many other fields related to linear programming, interior point methods do not seem to have made a big impact in the integer programming context. It is probably because the standard integer programming methodologies were originally proposed

at a time when the simplex method was the only efficient algorithm available for solving linear programming problems and, hence, they were biased to the features available in this method. Moreover, until a few years ago, interior point methods were not able to reoptimize a problem after carrying out modifications to the data as efficiently as a simplex type method, a scenario that has changed in the last years with the development of efficient warmstarting techniques for interior point methods (see Chapter 2, Section 2.3.1). Another reason may be due to previous unsuccessful attempts of straightforwardly replacing a simplex type method by an interior point method. These two methods are essentially different and, hence, should not be used in the same way.

In this chapter, we address the several facets of using the primal-dual interior point algorithm within a branch-price-and-cut method. We discuss in detail how to modify the core components of this method in order to exploit the advantageous features which are offered by the interior point algorithm. We believe it is an appropriate time for an investigation like this, as interior point methods have achieved a mature status concerning both theory and computational implementations. To verify the proposed approach, we present computational results for well-known instances of the vehicle routing problem with time windows (VRPTW). It is a classical integer programming problem that is widely used for testing new algorithms due to its difficulty. It is worth mentioning that the issues related to the integration of the interior point algorithm with the branch-price-and-cut method which we discuss here are not limited to or specialized for the VRPTW and can be straightforwardly used in other integer programming applications.

The remainder of this chapter is organized as follows. In Section 2, we present a literature review of previous attempts of combining interior point algorithms with integer programming methodologies. In Section 3, we address all the issues involved in the use of the primal-dual interior point algorithm in the branch-price-and-cut method and propose how to deal with them. The VRPTW is described in Section 4 and the results of computational experiments with the new approach solving the VRPTW instances are presented in Section 5. The conclusion and potential further developments are presented in Section 6.

5.1 Interior point methods and integer programming

Starting with Karmarkar's projective algorithm (Karmarkar, 1984), interior point methods have quickly and strongly evolved in the last few decades (Wright, 1997; Gondzio, 2012). They have been successfully applied not only to solving linear programming problems but also in many other areas such as quadratic, semi-definite, and conic programming. However, in spite of the close relationship between linear programming and integer programming, interior point methods have not showed a similar impact on integer programming. In this section we present a literature review of different attempts to use interior point methods within integer programming approaches that rely on linear programming relaxations. For a brief review on the basic concepts of the primal-dial interior point algorithm, please see Chapter 2.

To the best of our knowledge, the use of the primal-dual interior point algorithm within a full branch-price-and-cut framework has never been investigated in the literature. Moreover, few attempts have been presented in the literature regarding integer programming methodologies that are based on interior point algorithms. The first implementations in this sense started only in the beginning of the nineties, with the pioneering works by Mitchell and his collaborators. Mitchell and Todd (1992) combines the primal projective interior point algorithm with a cutting plane method and employed to solve the perfect matching problem. The authors propose to use early termination when solving the linear relaxations and present how to deal with several issues such as how to obtain a new iterate after adding constraints and

columns to a linear programming problem that has been just solved. As shown in the computational results, the proposed approach resulted in a reduction of the number of iterations and the number of calls to the separation subproblem, although the CPU times were not competitive with a cutting plane method based on the simplex method. The authors associate this behavior to the difficulty in calculating the projections required by the projective interior point method used in their experiments, as well as to the inefficient warm-starting strategy used after the addition of cutting planes. Indeed, a more efficient interior point cutting plane method was obtained a few years later, by using the primal-dual interior point method and improved warm-starting strategies (Mitchell and Borchers, 1996; Mitchell, 2000). According to the computational results presented for two integer programming applications, namely the linear ordering problem and the max-cut problem, the interior point approach was competitive with a cutting plane algorithm based on the simplex method.

The use of an interior point algorithm within a branch-and-bound method was first exploited by Borchers and Mitchell (1992). The authors observed that the primal-dual interior point algorithm tends to quickly find feasible solutions with good objective values, but then may spend a considerable amount of time approaching a solution accurate enough to meet the termination criteria. In addition, the iterates tend to converge steadily to an optimal solution and, hence, the authors propose to use an early branching strategy, which consist in not solving the node to optimality, but stopping as soon as it becomes apparent that the optimal solution of the node includes an integer variable at a fractional value. The authors present preliminary computational results on a set of mixed-integer programming problems. The interior point branch-and-bound was competitive in about half of the instances in comparison with a state-of-the-art branch-and-bound algorithm based on the simplex method. In the remaining instances, the latter dominates as the simplex method was significantly faster than the primal-dual interior point method in solving the LP relaxations of these problems. The authors associate the inferior performance of the new approach to the lack of efficient warm-start techniques for the primal-dual interior point method, a situation that has changed significantly since then.

du Merle et al. (1999) propose the use of the analytic center cutting plane method (ACCPM) within the branch-and-price method for solving the minimum sum-of-squares nonhierarchical clustering problem, a constrained hyperbolic program in 0-1 variables (see Aloise et al. (2012) for a recent investigation of this problem). In the ACCPM approach, an interior point method is used to obtain approximate analytic centers of the localization set, which corresponds to the dual feasible set with an additional inequality for the best bound found so far (see Goffin and Vial (2002) for further details of the method). According to the computational results for several fairly large data sets publicly available, the proposed combination is very promising and allows the exact resolution of substantially larger instances than those treated before. Recently, a similar combination was investigated in (Elhedhli and Goffin, 2004) for integer programming problems. The authors show how to modify the ACCPM so that it can warm-start after branching as well as after the addition of columns in the primal problem. In the computational experiments, the authors have used randomly generated instances of the bin packing problem and the capacitated facility location problem. The results show that for the bin packing instances, the branch-and-price method using the ACCPM results in a number of nodes that is comparable with the branch-and-price method using the standard column generation. However, the former requires on average 58% of the CPU time required by the latter. When comparing the two methods for the facility location instances, the method using ACCPM explores less nodes and requires less computational time. Moreover, it is consistently better than the method using the standard column generation in making fewer calls to the pricing subproblems and requiring less computational time per node.

5.2 Main issues of an interior point branch-price-and-cut method

In this section, we discuss the issues of using the primal-dual interior point algorithm within the branch-price-and-cut method. We address the main elements involved in this combination, namely the column generation technique, the separation and addition of valid inequalities, branching, and the use of primal heuristics. As it will be seen in the computational results, the effort to overcome the challenges pays off in a number of advantageous features offered by the new approach.

5.2.1 Primal-dual column generation

The column generation technique is an iterative method applied to solve a linear programming problem in which the coefficient matrix has a huge number of columns, but these columns can be generated by a rule that is known in advance. The idea is to keep only a subset of columns in the coefficient matrix, and reach an optimal solution without explicitly generating all the columns. In the column generation literature, the linear programming problem we are interested in is called the (continuous) master problem (MP). This problem has a huge number of variables and hence we recur to an auxiliary problem, which has only a subset of variables of the MP. This auxiliary problem is called the restricted master problem (RMP). Iteratively, the columns of the MP are generated and then added to the RMP by calling the oracle (or, pricing subproblem). For a full description of the column generation method, see Chapter 4.

The standard column generation is based on optimal solutions of the dual set. However, the use of these solutions causes instability in the column generation method, due to the high oscillation between extreme points of consecutive outer iterations. A slow progress is typically observed, specially during the last iterations of the method. To avoid this weakness of the method, different techniques have been proposed in the literature in order to obtain non-extremal dual points which lead to more stable strategies. In Chapter 4, we address the primal-dual column generation technique, which relies on the primal-dual interior point algorithm to solve the RMP, so that non-optimal, well-centered dual points are obtained, leading to a naturally stable strategy. Promising computational results were presented in that chapter for this technique when solving master problems obtained from linear-relaxed reformulations of three classical integer programming problems: the cutting stock problem, the vehicle routing problem with time windows, and the capacitated lot sizing problem with setup times. According to the results, the primal-dual column generation technique outperforms the standard column generation as well as the analytic center cutting plane method, considering the number of iterations and CPU time, on average, for all the applications used in the experiments. Moreover, the results show that the larger the instances, the better is the relative performance of the primal-dual approach. Therefore, we conclude that an improved column generation procedure was obtained by exploiting the advantages offered by the interior point algorithm. With this in mind, in the next section we investigate the use of the primal-dual interior point algorithm in the generation of valid inequalities, in particular for the branch-price-and-cut method.

5.2.2 Primal-dual column and cut generation

For some applications, valid inequalities can be used to improve the bounds provided by the master problems and consequently reduce the number of nodes in the search tree. Different types of valid inequalities are available in the literature and their effectiveness usually depends on the problem. Besides, in the context of column generation, the addition of valid inequalities

is not a trivial task as it may drastically increase the computational cost of solving the pricing subproblem and, hence, a good trade-off must be achieved (Desaulniers et al., 2011; Desrosiers and Lübbecke, 2010; Poggi de Aragão and Uchoa, 2003).

Consider the RMP in a given iteration of the column generation procedure (which may be the last one). Assume that the current primal solution is fractional so that a *separation procedure* can be used to generate a subset of violated valid inequalities. By adding these valid inequalities to the problem, we obtain

$$z'_{RMP} := \min \quad \sum_{j \in \overline{N}} c_j \lambda_j, \tag{5.1a}$$

$$\text{s.t.} \quad \sum_{j \in \overline{N}} a_j \lambda_j = b, \tag{5.1b}$$

$$\sum_{j \in \overline{N}} h_j \lambda_j \leq d, \tag{5.1c}$$

$$\lambda_j \geq 0, \qquad \forall j \in \overline{N}, \tag{5.1d}$$

where $d \in \mathbb{R}^{m'}$, $h_j \in \mathbb{R}^{m'}$, for all $j \in \overline{N}$, and $m' > 0$ is the number of valid inequalities. Let $\bar{\sigma} \in \mathbb{R}^{m'}$ be a dual solution associated to (5.1c). This dual solution must be taken into account in the pricing subproblem, which can be stated as

$$z'_{SP}(\overline{u}, \overline{\sigma}) := \min\{0; c_j - \overline{u}^T a_j - \overline{\sigma}^T h_j \mid j \in N\}, \tag{5.2}$$

where h_j may represent an intermediate variable vector defined as a function of a_j or as a function of other variables of the subproblem (see Spoorendonk (2008); Desaulniers et al. (2011) for further details). Depending on the strategy used to solve the subproblem as well as on the type of the valid inequalities, the subproblem (5.2) may become considerably more difficult to solve than (4.3). In such case, this difficulty typically increases as m' grows, so an important concern is to keep the number of valid inequalities small.

Separation procedures are usually called at the optimal solution of the master problem, *i.e.*, after the column generation procedure has finished. However, as it was already mentioned in Section 5.1, calling the separation procedure before reaching optimality is likely to result in deeper cuts, as well-centered points in the interior of the feasible set are used to generate the valid inequalities. Besides, this strategy facilitates the warm-start when an interior point method is used to solve the RMP, as discussed in Section 5.2.5. With these observations in mind, we propose to modify the PDCGM oracle presented in the previous section so that two operations are available. Either new columns are generated by calling the pricing subproblem, or new constraints are generated by calling the separation subproblem. The separation subproblem is in charge of generating valid inequalities for the RMP, based on the current primal (feasible) solution. A few drawbacks are associated to this early search for valid inequalities: (*i*) it may be too early to call the separation subproblem and it will only waste time without returning any violated inequality; (*ii*) the search may find too many valid inequalities, as an inequality may be violated by the current iterate but not by an optimal solution. As observed by Mitchell (2000), these disadvantages may be minimized by keeping the iterates well-centered in the feasible set, and by using a tolerance threshold to start generating valid inequalities, as discussed below. Furthermore, the gain of using the early search strategy is likely to overcome these potential drawbacks.

Algorithm 8 is an extension of primal-dual column generation algorithm (see Section 4.3), which takes the generation of valid inequalities into account. The oracle procedure was modified so that the primal solution of the RMP is also sent as a parameter to the oracle in line

7. This procedure is not detailed in the algorithm, as different strategies may be used to define how the pricing and the separation subproblems should be called. For instance, valid inequalities should be generated only after the relative gap falls below a tolerance threshold ε_c. In other words, the oracle should call the separation subproblem only if $gap < \varepsilon_c$. In practice, $\varepsilon_c = 0.1$ or $\varepsilon_c = 0.01$ are typically good choices. The value of ε_c may also be dynamically adjusted according to the maximum violation and the number of violated constraints, as suggested by Mitchell and Borchers (1996). In addition, it is important to avoid calling the separation subproblem in two consecutive outer iterations. By alternating between the two types of subproblems, more accurate points are sent to the oracle and a better overall performance is likely to be achieved in practice.

Algorithm 8: The Primal-Dual Column and Cut Generation Method.

Input: Initial RMP; parameters κ, $\varepsilon_{\max} > 0$, $D > 1$, $\delta > 0$.
Output: optimal solution λ.

```
1  Set LB = −∞, UB = ∞, gap = ∞, ε = 0.5;
2  While (gap ≥ δ) do
3      find a well-centered ε-optimal solution (λ̃, ũ) of the RMP;
4      if in the last iteration new cuts were added to the RMP then UB = c^T λ̃;
5      else UB = min(UB, c^T λ̃);
6      call ORACLE(λ̃, ũ, σ̃);
7      if new cuts were generated then add them to the RMP;
8      else LB = max(LB, κ z'_SP(ũ, σ̃) + b^T ũ);
9      gap = (UB − LB)/(1 + |UB|);
10     ε = min{ε_max, gap/D};
11     if (z'_SP(ũ, σ̃) < 0) then add the new columns to the RMP;
12 end(while)
```

Although the early search for valid inequalities has been exploited in cutting plane methods in the literature, the reader should notice that in the context addressed here, the separation procedure will be called in the course of the column generation algorithm. It is an interesting situation if we notice that we will cut-off part of the primal feasible set, which may be expanded again by generating new columns in the next call to the pricing subproblem. Moreover, as mentioned above, the subproblem may quickly become more and more difficult to solve as m' increases. Hence by reducing the number of valid inequalities due to the use of non-optimal, well centered solutions, we hope to improve the overall performance of the algorithm.

5.2.3 Branching

Branching is another core element of the branch-price-and-cut method. Given a fractional solution of the linear relaxation associated to a node, the branching procedure usually splits the node into two child nodes, each one having a partition of the feasible set. Different branching rules are available and their efficiency is typically problem-dependent. Furthermore, the standard branch-and-bound strategy of selecting only one fractional component to branch is usually inefficient and can be prohibitive in a branch-price-and-cut context, so more elaborate strategies must be used. On the other hand, it is essential to use a branching rule that is compatible with the column generation scheme, *i.e.*, a rule that does not increase too much the difficulty of solving the pricing subproblem (Villeneuve et al., 2005; Poggi de Aragão and Uchoa, 2003). For instance, as it will be discussed in Section 5.3.4, the branching rule used for the vehicle routing problem with time windows is imposed on the pricing subproblem, without compromising the performance of solving the subproblem. The initial RMP of a (child) node,

corresponds to the last RMP of its parent node without the columns that violate the new constraints imposed by the branching rule.

When solving the linear relaxation of a node by an interior point method, *early branching* is likely to result in a better overall performance of the branch-and-bound search (Borchers and Mitchell, 1992). Similarly to what is done in the column and cut generation, the early branching consists in calling the branching procedure before reaching the optimality of the linear relaxation (master problem). It is based on the fact that, when solving a standard linear programming problem, an interior point method gets close to optimality quickly but then may need a considerable effort to attain the required accuracy of the optimal solution. Indeed many components of the solution vector may reach the values which are very close to the optimum after merely a few interior point iterations. As showed by Borchers and Mitchell (1992), it can be useful in reducing the CPU time of the overall method. However, it is not clear whether the use of early branching in combination with column and cut generation is beneficial. In fact, the columns and cuts added in the very last iterations may be decisive for the values of the components of an optimal solution and, hence, the early branching may be misguided by wrong estimates of the solution values.

Here we propose to deal with the branching operation in two steps. In the first step, called preprocessing, we aim at quickly obtaining a non-optimal solution of the master problem of the current node so that the first evidence of a potentially good branching decision is taken from it. If the node is not eligible for pruning, then the branch is carried out and the second step is not necessary. Otherwise, the second step is started in which the master problem is solved to optimality. Different strategies may be used to solve the problem quickly, such as setting a loose optimality tolerance $\varepsilon_b < \varepsilon_c$ in the column and cut generation procedure, and using a heuristic method to solve the pricing subproblem as it is usually the most time-demanding procedure.

5.2.4 Primal heuristics

The use of quick heuristic methods is crucial in the branch-price-and-cut method, specially for identifying integer feasible solutions that can be used to prune nodes. These heuristics may be simple generic procedures that are based on rounding the components of a fractional solution, but also they can be very elaborate and exploit the structure of the problem. We are interested in general-purpose heuristics that can be used in combination with the column and cut generation. After solving the master problem using the strategy discussed in Section 5.2.2, we propose to call a primal heuristic that attempts to quickly identify an integer feasible solution. The use of rounding heuristics within the column generation is usually very helpful (Vanderbeck, 2000; Degraeve and Peeters, 2003), as the columns correspond to parts of an integer solution of the original problem. Hence, by putting together a feasible subset of them, an integer solution is likely to be obtained. For instance, in the VRPTW a column corresponds to a feasible route for a vehicle. By selecting a subset of routes (columns) such that each customer is visited exactly once by these routes, we obtain a feasible integer solution. In the approach presented in this chapter, we rely on a residual rounding heuristic, described as follows.

Consider we have a fractional primal solution $\tilde{\lambda}$. The first step is to select up to T components of $\tilde{\lambda}$ and round them to an integer value. Then, the columns corresponding to the rounded components are multiplied by the new values and subtracted from b, the right-hand side vector of the RMP. The resulting problem, called residual RMP, is solved again and the process is repeated until an integer solution is obtained or it is detected that no feasible integer solution can be found by the heuristic. It may happen that the residual RMP becomes

infeasible and hence, the column generation must be called again until a feasible solution is obtained or the infeasibility is also identified by the column generation procedure. All the columns associated to the rounded components should be stored to eventually give rise to an integer feasible solution. Even though a modified RMP is solved inside the heuristic, any column that is generated during its course is also valid for the original RMP.

The rounding procedure is detailed in Algorithm 9. Ideally, the parameter T (line 5) should be very small (*e.g.*, $T = 2$), but notice that this is likely to result in a large number of iterations of the heuristic and, hence, a good compromise must be found. In line 6, different strategies could be used to select an index j. For example, j can be chosen so that λ_j is the value with the fractional part closest to 0.5. Different strategies can be used in line 9 as well. Given a fractional value $\tilde{\lambda}_j$ it may be rounded to its nearest integer, or rounded-up to the smallest integer greater than or equal to its value (*i.e.* $\lceil \tilde{\lambda}_j \rceil$), or still rounded-down to the largest integer less than or equal to its value (*i.e.* $\lfloor \tilde{\lambda}_j \rfloor$). In all cases, the choice of the best strategy depends on the problem we are dealing with.

Algorithm 9: Residual rounding heuristic.

Input: RMP, solution $\tilde{\lambda}$, tolerance $\tilde{\varepsilon}$; parameters $T > 0$.
Output: A set of column indices $\mathcal{T}$ which leads to an integer solution; Fail, otherwise.

```
1  Set b̃ = b, T = ∅, N = N̄;
2  Initialize r as the number of constraints in the RMP that are violated for λ = 0;
3  while r > 0 do
4      set t = 0;
5      while t < T and |N| > 0 do
6          select an index j ∈ N;
7          if λ̃_j > 0 then
8              add j to T and increment t;
9              round λ̃_j to obtain the integer value [λ̃_j];
10             reset the right-hand side of the RMP as b̃ := b̃ − [λ̃_j]a_j;
11             solve the resulting RMP with tolerance ε̃ to obtain a new λ̃;
12             if the RMP became infeasible then call Algorithm 8 with δ = ε̃;
13             if Algorithm 8 has detected the RMP is infeasible then STOP;
14         end(if)
15         remove j from N;
16     end(while).
17 end(while).
18 if r = 0 then T leads to a feasible integer solution of the RMP.
```

5.2.5 Warmstarting strategy

The core elements of a branch-price-and-cut method share a common feature: they typically involve the solution of a linear programming problem that is a simple modification of another linear programming problem that has been already solved, *i.e.*, these problems are *closely-related*. For instance, after solving an RMP in a given iteration of the column and cut generation procedure, the RMP to be solved in the next iteration differs from the previous one by having either new columns or new constraints in the primal formulation. Any information available from the solution of the previous RMP may be useful to speed up the solution of the new one. A technique that is able to properly exploit such information and obtain an initial solution of the problem with beneficial characteristics in relation to a default initial solution is called a *warmstarting technique*. See Chapter 2 for a brief review on the use of this techniques within interior point methods.

In the interior point branch-price-and-cut method which is proposed in this study, we follow the warmstarting strategies by Gondzio (1998) and Gondzio and Grothey (2003). More specifically, the RMPs are solved approximately with loose accuracy requirements. Non-optimal and well-centered solutions are stored and they are later used to obtain warmstarting points for that new problems that are created in the primal-dual column and cut generation process. The solution $(\tilde{\lambda}, \tilde{u})$ sent to the oracle (see Section 5.2.2) is the same as the one stored for constructing a warmstarting point to be used in the next outer iteration, except when the relative gap becomes less than a threshold value ε_{ws} (*e.g.*, $\varepsilon_{ws} = 10^{-3}$) which means the solution is too close to the boundary for leading to a good warmstart. In this particular case, we store the iterate when for the first time the relative gap falls below ε_{ws} and continue solving the problem until the predefined optimality tolerance is reached. Before solving the next RMP, the stored point is modified to reduce the primal and dual infeasibilities, and multiple centrality correctors (Gondzio, 1996; Colombo and Gondzio, 2008) may be applied to improve the centrality of the warmstarting point.

5.3 The vehicle routing problem with time windows (VRPTW)

The VRPTW is a widely studied integer programming problem and covers a broad range of real-world applications. All the time, many companies over the world are looking for the best routes for delivering/collecting products to/from customers, which are usually spread in a certain neighborhood, city or even larger regions. The VRPTW can be used to model these situations and the efficient solution of real-life problems is crucial for many businesses – for interesting examples, see e.g. Braysy and Gendreau (2005); Desaulniers et al. (2010); Pureza et al. (2012); Macedo et al. (2011). Moreover, the VRPTW is important as a benchmarking for testing new solution strategies, because it is considered a very difficult integer programming problem.

5.3.1 Extended formulation

In the standard VRPTW, a set of vehicles is available in a single depot and these vehicles must be used to visit a set of customers $C = \{1, \ldots, m\}$ in order to satisfy the demands d_i of each customer $i \in C$. The problem consists in determining a set of minimum cost routes for the vehicles, satisfying the following requirements. Each customer must be visited exactly once and the arrival of vehicle must satisfy a time window $[w_i^a, w_i^b]$, *i.e.*, the vehicle cannot arrive after time w_i^b to service a customer, and it should wait until time w_i^a in case it arrives too early. We denote by s_i the service time for customer i, and by t_{ik} the travel time from customer i to another customer k. The number of customers a vehicle can serve is limited by its maximum capacity q, and the vehicle must return to the depot after visiting the customers covered by its route. We assume the vehicles are identical and they are available in a sufficient number to service all the customers. The cost of a route is given by the total distance traveled by the vehicle. For a detailed description of the problem, see Appendix A.

Although a compact formulation of the VRPTW has been proposed in the literature, its linear relaxation provides a poor bound for the value of an optimal integer solution. As a consequence, a pure branch-and-bound strategy or even a branch-and-cut method are usually inefficient in practice. A better bound can be obtained by recurring to the linear relaxation of an extended formulation of the VRPTW, given by the following integer master problem with a set-partitioning structure

$$\min \quad \sum_{j \in N} c_j \lambda_j \tag{5.3a}$$

$$\text{s.t.} \quad \sum_{j \in N} a_j \lambda_j = 1, \tag{5.3b}$$

$$\lambda_j \in \{0,1\}, \qquad \forall j \in N, \tag{5.3c}$$

where $a_j = (a_{1j}, \ldots, a_{mj})^T$ is a column constructed from a feasible route, c_j is the corresponding cost to visit each customer in the route and then come back to the depot, and N is the set of indices of all feasible routes for the problem. The coefficients in column a_j are given by $a_{ij} = 1$ if route j visits customer i and $a_{ij} = 0$ otherwise.

Formulation (5.3) is solved by a branch-and-price strategy. At each node, the integrality of the master variable λ is dropped and the resulting linear programming relaxation is solved by the column generation technique. To generate columns, feasible routes are obtained by solving an elementary shortest path problem with resource constraints (ESPPRC). Consider that a subset of columns of the MP was obtained in previous iterations. Assume a primal-dual feasible solution $(\tilde{\lambda}, \tilde{u})$ of the corresponding RMP is available. The dual solution is used to obtain one or more routes by solving the following subproblem:

$$\min \quad \sum_{i \in C_0} \sum_{k \in C_0} (c_{ik} - \tilde{u}_k) x_{ik} \tag{5.4a}$$

$$\text{s.t.} \quad \sum_{k \in C_0} x_{0k} = 1, \tag{5.4b}$$

$$\sum_{i \in C_0} x_{ih} - \sum_{k \in C_0} x_{hk} = 0, \qquad \forall h \in C, \tag{5.4c}$$

$$\sum_{i \in C_0} x_{i,m+1} = 1, \tag{5.4d}$$

$$\sum_{i \in C} d_i \sum_{k \in C_0} x_{ik} \leq q, \tag{5.4e}$$

$$w_i + s_i + t_{ik} - M(1 - x_{ik}) \leq w_k, \qquad \forall\, i,k \in C_0, \tag{5.4f}$$

$$w_i^a \leq w_i \leq w_i^b, \qquad \forall i \in C_0, \tag{5.4g}$$

$$x_{ik} \in \{0,1\}, \qquad \forall\, i,k \in C_0, \tag{5.4h}$$

where x_{ik} is a binary variable that represents whether the vehicle goes directly from customer i to customer k, w_i is a real variable that gives the time customer i is visited, $C_0 = C \cup \{0, m+1\}$ is a set of customer indices with 0 and $m+1$ representing the depot, and M is a sufficiently large number. Constraints (5.4b)-(5.4d) guarantee that the vehicle leaves the depot and comes back to it in the end of the route, and also that the vehicle always leaves a customer once it is visited. The maximum capacity of the vehicle is imposed by constraint (5.4e), and the time windows of the customers are defined in constraints (5.4f)-(5.4g). Given a solution $[x_{ik}^\star, w_i^\star]_{i,k \in N}$ of problem (5.4), a new column a_j is generated by setting

$$a_{ij} = \sum_{k \in C_0} x_{ik}^\star, \quad \forall i \in C.$$

Both an optimal solution of (5.4) and any suboptimal solution with a negative objective value can be used to generate a column. In practice, the ESPPRC is solved by using a dynamic programming strategy, as described in Section 5.3.2.

The first investigations concerning the use of a branch-and-price method for the VRPTW were presented by Desrosiers et al. (1984) and Desrochers et al. (1992). Since then several improvements have been proposed in this context, as a result of intensive investigation in the area (see Irnich and Desaulniers, 2005; Kallehauge et al., 2006; Feillet, 2010; Baldacci et

al., 2012; Rousseau et al., 2007). Branch-price-and-cut methods for the VRPTW were first proposed by Kohl et al. (1999) and Cook and Rich (1999), using the so-called two-path and k-path valid inequalities. Recently, the subset row inequalities and the generalized k-path inequalities were proposed by Jepsen et al. (2008) and Desaulniers et al. (2008), respectively, leading to more powerful methods. The subset row inequalities are described in Section 5.3.3. They are used in our implementation due to their superior overall performance in practice.

5.3.2 Solving the ESPPRC

The most successful implementations for solving the ESPPRC are based on a dynamic programming algorithm. More specifically, it is a label-setting algorithm that was proposed by Desrochers (1988) and Beasley and Christofides (1989), and has been significantly improved in the last years by the development of different techniques, such as the identification of unreachable nodes (Feillet et al., 2004) and the bidirectional label extension with resource bounding (Righini and Salani, 2008). Also, heuristics and meta-heuristics have been used to speed up the implementations (Chabrier, 2006; Desaulniers et al., 2008). In this section, we describe the label-setting algorithm we have implemented to solve the ESPPRC. It is based on the aforementioned publications and is presented here in order to have a self-contained description that covers all those contributions in a unified way. The use of the subset row inequalities introduces a slight modification of the algorithm described in this section, as it will be seen in Section 5.3.3.

Let $\mathcal{G}(\mathcal{V}, \mathcal{A})$ be a graph defined by the set $\mathcal{V}$ of vertices associated to indices in C_0 (the set of all customers plus the depot, with 0 as the source vertex and $m+1$ as the sink vertex). A vertex $i \in \mathcal{V}$ is adjacent to a vertex $k \in \mathcal{V}$ if arc (i,k) belongs to the set of arcs, denoted by $\mathcal{A}$. To each arc (i,k) is associated the cost $c_{ik} - \tilde{u}_k$, similarly to (5.4a). In the column generation context, $\tilde{u}_1, \ldots, \tilde{u}_m$ are the dual variables associated to master constraints, and we define $\tilde{u}_0 := 0$ and $\tilde{u}_{m+1} := 0$. In the general context, they can be seen as prizes collected when the respective vertex is visited and, under this point of view, c_{ik} becomes the actual cost of traversing arc (i,k).

In the label-setting algorithm, each route is obtained by iteratively extending labels which represent partial paths. A label L_i consists in a set of attributes that characterize the partial path that ends in vertex $i \in \mathcal{V}$. The attributes considered here are: $\bar{c}(L_i)$, the total cost along the path; $q(L_i)$, the load accumulated along the path; $t(L_i)$, the total time spent in the path; and $v_k(L_i)$, a flag that indicates whether vertex k is visited or not by the path, for all $k \in \mathcal{V}$. By the feasible *extension* of a label, new vertices are added to the corresponding path. Here, we use the bidirectional label-setting strategy, which means that labels are generated by forward as well as backward extension. Recall that t_{ik} denotes the travel time between vertices i and k, s_i is the service time and d_i is the demand of vertex i. In the forward extension, any partial path starts at the source and it is extended until the time attribute becomes greater than or equal to $T/2$, where T is the maximum feasible arrival time at the sink, which is given by the largest value $w_l^b + s_l + t_{l,m+1}$ over all $l \in \mathcal{V}$. Given a label F_i, $i \in \mathcal{V}$, we extend it to each vertex $k \in \mathcal{V}$ such that $(i,k) \in \mathcal{A}$, $v_k(F_i) = 0$, and the attribute values of the resulting label F_k satisfy $q(F_k) \leq q$ and $t(F_k) \leq w_i^b$. The attributes of the resulting label are given by

$$\begin{aligned}
\bar{c}(F_k) &= \bar{c}(F_i) + c_{ik} - \tilde{u}_k, \\
q(F_k) &= q(F_i) + d_k, \\
t(F_k) &= \max\{w_k^a,\ t(F_i) + s_i + t_{ik}\}, \\
v_k(F_k) &= 1,
\end{aligned}$$

$$v_l(F_k) \;=\; v_l(F_i), \;\; \forall l \neq k.$$

The source label F_0 is initialized with all attribute values equal to zero, except for $v_0(F_0) = 1$. The paths in the backward extension start at the sink and the labels are extended until the time attribute becomes greater than or equal to $T/2$. Given a label B_i, $i \in \mathcal{V}$, we extend it to each vertex $k \in \mathcal{V}$ such that $(k, i) \in \mathcal{A}$, $v_k(B_i) = 0$, and the attribute values of the resulting label B_k satisfy $q(B_k) \leq q$ and $t(B_k) \leq T - w_i^a$. The extended label has the attributes

$$\begin{aligned}
\bar{c}(B_k) &= \bar{c}(B_i) + c_{ik} - \tilde{u}_k, \\
q(B_k) &= q(B_i) + d_k, \\
t(B_k) &= \max\{T - (w_k^b + s_k),\; t(B_i) + s_i + t_{ik}\}, \\
v_k(B_k) &= 1, \\
v_l(B_k) &= v_l(B_i), \;\; \forall l \neq k.
\end{aligned}$$

The sink label B_{m+1} is initialized with all of the attribute values equal to zero, except for $v_{m+1}(B_{m+1}) = 1$. In both types of extension, we keep a list of labels which are sorted in ascending order by the time attribute value. At each iteration, the first label of the list is selected, and the label-setting stops when the list becomes empty. We try to extend a label of vertex i to each adjacent vertex $k \in \mathcal{V}$, following the ascending order of the cost $c_{ik} - \lambda_k$. It is important to check for *unreachable* vertices after creating a label. A vertex $l \in \mathcal{V}$ is said unreachable for a label L_k when $q(L_k) + d_l > q$ or $t(L_k) + s_k + t_{kl} > w_l^b$. If some of these inequalities are satisfied we modify the visiting attribute value by setting $v_l(F_k) := -1$, so this information can be used later in the algorithm. *Dominance rules* should also be used in order to reduce the number of labels created in the course of the algorithm. They are based on discarding labels that are guaranteed not to lead to an optimal route. Given two labels L_i and L_i' associated to a vertex $i \in \mathcal{V}$, L_i does not dominate L_i' if at least one of the following conditions are satisfied

$$\bar{c}(L_i) \;>\; \bar{c}(L_i'), \tag{5.5}$$

$$q(L_i) \;>\; q(L_i'), \tag{5.6}$$

$$t(L_i) \;>\; t(L_i'), \tag{5.7}$$

$$v_l(L_i) \;\neq\; 0 \;\text{ and }\; v_l(L_i') \;=\; 0, \;\text{ for some } l \in \mathcal{V}. \tag{5.8}$$

In such case, L_i' must be further extended in the algorithm, as it can result in an optimal route. Otherwise, L_i' is dominated by L_i and can be discarded without compromising the optimal solution. After both forward and backward extensions have finished, each forward label F_i is linked to each backward label B_i, for all $i \in \mathcal{V}$, if it results in a feasible route with a total cost $\bar{c}(F_i) + \bar{c}(B_i)$ that is negative. The route with the least total cost is the optimal solution of the ESPPRC. The remaining routes obtained by the algorithm are suboptimal solutions that can also be used to generate columns for the restricted master problem.

The label-setting algorithm is initialized by the set of routes associated to the columns in the restricted master problem. Up to K_1 columns are used and we select those associated to the smallest values s_j, the slack dual component related to λ_j, for all $j \in \overline{N}$. We also use an adaptation of the Clarke and Wright heuristic (Clarke and Wright, 1964; Braysy and Gendreau, 2005) to construct initial routes that are inserted into this set. Furthermore, three improvement heuristics are applied upon the best paths in the set. The first one consists in, given a route, removing one customer at a time from it. In the second heuristic, we try to insert one customer that is not in the route, between each two customers in the route. The third heuristic is given by swapping any two consecutive customers of a route. In all methods,

we try to improve up to K_2 routes of the initial set, those with the least reduced costs. Each heuristic is called once and a modified route is inserted into the set if it has a negative reduced cost.

As proved by Dror (1994), the ESPPRC is a strongly NP-hard problem. Finding the optimal route at each call to the oracle is computationally expensive and should be avoided in the column generation procedure, as the optimal solution is needed only at the last outer iteration. This way, a few simplifications are made on the exact label-setting algorithm, so that it becomes a relatively quick heuristic. One simplification is given by imposing a limit K_d on the number of vertices for which we verify conditions (5.8) in the dominance checking. Besides, a given label is extended to at most K_e adjacent vertices. Another simplification consists in setting a limit K_l for the total number of labels generated by extensions. The values of these three parameters are dynamically chosen according to the relative gap in the column generation procedure. When the relative gap falls below the column generation optimality tolerance δ, we discard these simplifications and call the exact label-setting algorithm.

The first branch-and-price implementations for the VRPTW were based on the SPPRC, a relaxation of the ESPPRC that allows the routes to be non-elementary, *i.e.*, it allows cycles (Desrosiers et al., 1984; Desrochers et al., 1992). The advantage is that the SPPRC can be solved in pseudo-polynomial time using a label-setting algorithm similar to the one just described in this section. However, the lower bounds obtained at the end of the column generation procedure may be worse than those obtained when the ESPPRC is used as the pricing subproblem, which may adversely affect the performance of the branch-and-bound search. To improve those bounds, a 2-cycle elimination is proposed by Desrochers et al. (1992) and extended to k-cycle elimination by Irnich and Villeneuve (2006). Even though, the current state-of-the-art implementations suggest that using the ESPPRC as the pricing subproblem results in a better overall performance of the branch-and-price method.

5.3.3 Valid inequalities for the VRPTW

Different types of valid inequalities are available for the VRPTW. Some of them are based on the variables of the compact formulation of the problem (Kohl et al., 1999; Cook and Rich, 1999; Desaulniers et al., 2008), which do not increase the difficulty of solving the pricing subproblem in a branch-and-price approach. Others are based on the extended formulation (Jepsen et al., 2008; Spoorendonk, 2008), which leads to tighter bounds at the cost of having a more difficult pricing subproblem. In our implementation, we use a valid inequality of this second type, namely the *subset row* (SR) inequality proposed by Jepsen et al. (Jepsen et al., 2008). This inequality is a Chvatal-Gomory rank-1 cut inspired by the clique and odd-hole inequalities for the set partitioning formulation. In the remainder of this section we briefly describe the SR inequality and discuss the issues involved in adding the corresponding dual variables to the subproblem.

Consider an RMP associated to problem (5.3). Given a subset $S \subset C$ and a scalar θ such that $0 < \theta \leq |S|$, the corresponding SR inequality is defined as

$$\sum_{j \in \overline{N}} \left\lfloor \frac{1}{\theta} \sum_{i \in S} a_{ij} \right\rfloor \lambda_j \leq \left\lfloor \frac{|S|}{\theta} \right\rfloor, \tag{5.9}$$

where $\overline{N} \subset N$ is the set of indices of master variables in the RMP. Such inequality is valid for any problem having the set-partitioning structure and follows from Chvatal-Gomory's procedure. In fact, for any subset S, if we take for each $i \in S$ the master constraints (5.3b),

scale each one with $1/\theta$ and add them all, the resulting inequality is

$$\sum_{i \in S} \frac{1}{\theta} \sum_{j \in \overline{N}} a_{ij} \lambda_j \leq \frac{|S|}{\theta}.$$

Now, taking the floor on both sides we obtain the SR inequality (5.9). As proved in (Jepsen et al., 2008), the separation problem of SR inequalities is *NP*-complete. In the same paper, the authors suggest the choices $|S| = 3$ and $\theta = 2$, as SR inequalities using other values seldom appeared in the computational experiments carried out on benchmarking instances. In this particular case, the SR inequality for a given subset S is given by

$$\sum_{j \in \overline{N}_S} \lambda_j \leq 1,$$

where $\overline{N}_S \subset \overline{N}$ is the subset of routes which visit at least $\theta = 2$ customers that belongs to S.

In a given iteration of the column and cut generation algorithm presented in Section 5.2.2, let (5.1) represent the associated RMP. The dual variables $\sigma_1, \ldots, \sigma_{m'}$ associated to each valid inequality in this problem must be taken into account in the subproblem solver. It can significantly increase the difficulty of solving the subproblem, in particular when the label-setting algorithm is used for solving the ESPPRC. To reduce these effects, a modified dominance criteria for the label-setting algorithm is proposed by Jepsen et al. (2008) and described as follows. Extending the description presented in Section 5.3.2, we define for any label L_i the additional attributes $r_S(L_i)$, for each set S associated to an SR inequality in the current RMP. For a given set S, $r_S(L_i)$ is a counter for the number of customers in S that are visited by the partial path represented by label L_i. The attribute cost of the label must also be modified to consider the dual component σ_S but only if $r_S(L_i) \geq 2$, *i.e.*, the path visits at least two customers that belong to S. The cost attribute is initialized as zero on depot vertices, and after extending L_i to obtaining a label L_k, the cost is now given by the expression

$$\bar{c}(L_k) \quad = \quad \bar{c}(L_i) + c_{ik} - \tilde{u}_k - \sum_{S \in \mathcal{S}_1} \sigma_S,$$

in both forward and backward extensions, where $\mathcal{S}_1 = \{S \mid k \in S \text{ and } r_S(L_k) = 2\}$. In the dominance criteria, the condition (5.5) is replaced by

$$\bar{c}(L_i) - \sum_{S \in \mathcal{S}_2} \sigma_S > \bar{c}(L'_i),$$

where $\mathcal{S}_2 = \{S \mid \sigma_S < 0 \text{ and } r_S(L_i) \mod 2 > r_S(L'_i) \mod 2\}$.

In order to keep the number of valid inequalities small, a few requirements can be imposed in the separation subproblem. Similarly to Desaulniers et al. (2008), we accept only the inequalities which are violated by at least a predefined threshold ε_v. These inequalities are sorted in descending order of their violation and, then, the first K_v inequalities are added to the RMP. A customer is allowed to belong to at most K_s of the sets that are used to generate the valid inequalities. Moreover, as proposed in Section 5.2.2, the separation subproblem is called only after the relative gap in the column and cut generation procedure falls below the threshold value ε_c.

5.3.4 Branching on the VRPTW

Different branching rules have been proposed for branch-and-price methods for solving the VRPTW. In our developments, we follow the branching scheme used by Desaulniers et al. (2008). For the completeness of the presentation, the scheme is briefly described below. Suppose the master problem has been solved and a fractional solution $\tilde{\lambda}$ has been obtained. This solution can be expressed in terms of the variables in the compact formulation by using the equalities

$$\tilde{x}_{ik} = \sum_{j \in \overline{N}} p_{ik}^{j} \tilde{\lambda}_j, \qquad \forall\ i,k \in C_0$$

where p_{ik}^{j} is equal to 1 if the path associated to column j visits customer i and goes directly to customer k, and equal to 0 otherwise. Notice that in a feasible integer solution we have $x_{ik} \in \{0,1\}$, for all $i,k \in C_0$, as each customer can be visited only once. In the fractional solution, we select the most fractional component $\tilde{x}_{ik} \notin \{0,1\}$ and create two child nodes. By most fractional, we mean the component with fractional part that is closest to 0.5. On the left node, we modify the pricing subproblem by fixing $x_{ik} = 0$. In the label-setting algorithm, it is done by removing arc (i,k) from the set of arcs $\mathcal{A}$. On the right node, we fix $x_{ik} = 1$ which corresponds to removing from $\mathcal{A}$ every arc (i,l) such that $l \neq k$ and $l \neq m+1$, and every arc (l,k) such that $l \neq i$ and $l \neq 0$. In both branches, the initial RMP contains the columns in the last RMP of the parent node that satisfy the new constraint imposed on x_{ik}.

The search tree is exploited by the best-first strategy. Moreover, we use the two-step procedure described in Section 5.2.3. In the first step, the column and cut generation algorithm is stopped with a loose optimality tolerance $\varepsilon_b > \delta$ and all pricing subproblems are solved by the heuristic label-setting algorithm, *i.e.*, with the simplifications described in Section 5.3.2. In case it is not possible to branch, we go to a second phase in which the default optimality tolerance δ is adopted and the exact label-setting algorithm is used when the relative gap falls below δ.

5.4 Computational results

In this section, we present the results of computational experiments using the interior point branch-price-and-cut method proposed in the previous sections. For brevity we will refer to our implementation as IPBPC. We have selected a classical integer programming problem that is widely studied in the branch-and-price literature, the vehicle routing problem with time windows (VRPTW). Of course, having computational experiments for more classes of problems would be very interesting but the difficulty of implementing a branch-price-and-cut method led us to choose one application. Nevertheless, the VRPTW has been used in many previous researches as a benchmark for illustrating and testing new ideas and algorithms. Moreover, the strategies proposed in Section 5.2 are independent of the problem.

The experiments are run on the benchmarking instances proposed by Solomon (1987). These instances were defined about 30 years ago, but the most difficult ones were solved only in the last few years. They are classified according to the spatial distribution of customers: C instances follow a clustered distribution, R instances follow a random distribution, and RC instances follow a mix of both distributions. Following Desaulniers et al. (2008), we focus on the 56 largest instances in the set, all those having 100 customers. These instances are commonly presented in two distinct sets, namely 100-series and 200-series sets, and those in the second set have wider time windows and larger vehicle capacity, so they are more challenging for the pricing subproblem.

The IPBPC framework has been implemented in the C programming language. It relies on the primal-dual interior point method implemented in the HOPDM code (Gondzio, 1995, 2012) to solve all the linear programming problems. The procedures for solving the pricing and the separation subproblems have also been implemented and they follow the descriptions presented in Sections 5.3.2 and 5.3.3. Table 5.1 shows the default parameter settings adopted in the experiments presented in Section 5.4.1. For each parameter in the first column, the second column gives the section in which the parameter is defined, and the third column shows its default value. In Section 5.4.2 we run additional experiments to verify the impact of some of these parameter choices. The experiments were performed on a Linux PC with an Intel Core i7 2.8 GHz CPU and 8.0 GB of memory.

Parameter	**Section**	**Value**
δ	5.2.1	10^{-6}
$\varepsilon_{\max}$	5.2.1	1.0
ε_c	5.2.2, 5.3.3	0.1
ε_b	5.2.3, 5.3.4	0.001
K_1	5.3.2	100
K_2	5.3.2	100
K_d	5.3.2	0, if $gap > 0.1$
		30, if $0.1 \geq gap > 0.01$
		100, if $0.01 \geq gap$
K_e	5.3.2	30
K_l	5.3.2	3 x 10^4, if $gap > 0.1$
		4 x 10^4, if $0.1 \geq gap > 0.01$
		7 x 10^4, if $0.01 \geq gap > 0.001$
		10^5, if $0.001 \geq gap$
K_s	5.3.3	1
K_v	5.3.3	3
ε_v	5.3.3	0.05

Table 5.1: Parameter choices in the IPBPC implementation for the VRPTW

5.4.1 Best results and comparison with a simplex-based approach

Tables 5.2 and 5.3 show the results for the Solomon's instances in sets 100-series and 200-series, respectively, using the IPBPC with all the features discussed in Sections 5.2 and 5.3. In both tables, the first two columns show the name of the instance and the value of the optimal solution obtained by the IPBPC, respectively. In the remaining columns are given the total number of generated columns, the number of generated cuts, the number of nodes in the branch-and-bound tree, the total CPU time spent on the oracle, the total CPU time spent on solving the RMPs, and the total CPU time spent on solving the instance. All the CPU times are given in seconds.

The analysis of these results reveals that the oracle dominates the computational effort, a usual behavior in integer programming problems. Each instance in the 100-series set was solved to optimality in less than 2300 seconds. In the 200-series set, 5 instances could not be solved to optimality due to memory overflow in the pricing subproblem solver called at the last outer iteration of column generation (exact label-setting algorithm). These instances are the same ones that could not be solved by Desaulniers et al. (2008) and, hence, it was an expected behavior as the same strategy was used for solving the pricing subproblem in that implementation. Nevertheless, the value of the best integer solution found for each instance is shown in Table 5.3. For the instances solved to optimality, none of them required more than 17000 seconds.

Instance	Optimal	Columns	Cuts	Nodes	Oracle time	RMP time	Total time
C101	827.3	2165	0	1	0.83	0.28	1.29
C102	827.3	3105	0	1	1.45	0.50	2.20
C103	826.3	3403	0	1	2.95	0.75	3.98
C104	822.9	4280	0	1	7.61	1.29	9.21
C105	827.3	2452	0	1	0.94	0.34	1.48
C106	827.3	2882	0	1	1.21	0.42	1.87
C107	827.3	2734	0	1	1.16	0.40	1.77
C108	827.3	3182	0	1	1.95	0.62	2.84
C109	827.3	3303	0	1	2.20	0.68	3.16
RC101	1619.8	2466	35	1	1.83	1.26	3.23
RC102	1457.4	3923	84	1	7.94	5.58	13.95
RC103	1258.0	5150	130	3	75.97	16.44	94.46
RC104	1132.3	6337	214	9	2230.29	52.02	2292.06
RC105	1513.7	3060	32	1	2.71	1.52	4.40
RC106	1372.7	9896	539	61	634.69	134.59	855.87
RC107	1207.8	3877	60	1	20.60	4.37	25.30
RC108	1114.2	5019	97	1	170.36	11.96	182.71
R101	1637.7	1708	4	3	0.94	0.42	1.69
R102	1466.6	2372	0	1	1.26	0.35	1.74
R103	1208.7	3337	18	1	4.10	1.74	6.18
R104	971.5	5801	184	5	364.24	33.93	411.50
R105	1355.3	2820	43	3	3.02	2.33	6.24
R106	1234.6	3722	52	3	7.77	4.10	13.17
R107	1064.6	4671	127	3	61.03	15.06	78.09
R108	932.1	5548	165	1	391.37	24.66	416.64
R109	1146.9	7010	339	35	234.72	72.51	348.13
R110	1068.0	4506	121	3	44.38	14.46	62.04
R111	1048.7	18186	672	107	1336.66	206.39	1701.96
R112	948.6	7857	415	17	1403.98	127.79	1573.75
Total		134772	3331	269	7018.14	736.73	8120.90

Table 5.2: IPBPC results for the 100-series Solomon's instances.

It is useful to compare the performance of the IPBPC with a state-of-the-art branch-price-and-cut method for the VRPTW that uses the simplex method to solve the linear relaxations. Such comparison allows us to verify if the use of an interior point algorithm within a branch-price-and-cut method is indeed worthwhile. Hence, in Tables 5.4 and 5.5 we show the best results available in the literature for a simplex-based branch-price-and-cut method (Desaulniers et al., 2008), and a comparison with the results shown in Tables 5.2 and 5.3. The simplex-based implementation uses the same type of valid inequalities as presented in Section 5.3.3 and the pricing subproblem is solved using a procedure very similar to that described in Section 5.3.2. However, the results in Desaulniers et al. (2008) were run on a different computer, namely a Linux PC with a Dual Core AMD Opteron 2.6 GHz CPU, and hence the conclusions about CPU time should be taken cautiously. The first few columns in Tables 5.4 and 5.5 give the number of cuts, number of nodes and total CPU times, as presented by Desaulniers et al. (2008). The remaining columns give the ratio between the values in the first columns and the corresponding values in Tables 5.2 and 5.3. Only the instances that were successfully solved by both approaches are presented in the tables.

For each 100-series instance, the IPBPC generated not only a smaller (or equal) number of valid inequalities, but also a smaller (or equal) number of nodes when compared with the simplex-based BPC. In total, the simplex-based BPC required around 68% more valid inequalities and 31% more nodes. The IPBPC was about 5 times faster than the other approach, but the reader should be warned that the two approaches were run on different computers (we did not have access to the proprietary code used by Desaulniers et al. (2008)). Con-

Instance	Optimal	Columns	Cuts	Nodes	Oracle time	RMP time	Total time
C201	589.1	2718	0	1	5.07	1.52	7.02
C202	589.1	5766	0	1	11.25	5.66	17.37
C203	588.7	7552	0	1	29.02	10.96	40.64
C204	588.1	11451	0	1	710.92	29.63	741.33
C205	586.4	3344	0	1	5.05	2.01	7.41
C206	586.0	5346	0	1	8.30	4.48	13.22
C207	585.8	6230	0	1	13.76	6.59	20.90
C208	585.8	4884	0	1	12.17	4.91	17.49
RC201	1261.8	9009	17	3	23.58	14.75	42.27
RC202	1092.3	9648	9	1	34.43	17.34	53.15
RC203	923.7	12238	6	2	305.02	34.56	341.50
RC204	783.5*						
RC205	1154.0	8250	9	1	24.68	11.37	36.69
RC206	1051.1	9430	9	1	61.51	22.00	84.27
RC207	962.9	14622	116	3	2487.92	190.59	2691.81
RC208	776.1*						
R201	1143.2	8466	20	1	21.05	14.58	36.24
R202	1029.6	12461	62	9	89.49	51.79	147.21
R203	870.8	14117	18	1	295.70	44.32	342.07
R204	731.3*						
R205	949.8	17262	112	13	738.09	193.86	968.18
R206	875.9	18307	81	3	2042.07	113.64	2166.28
R207	794.0	15090	9	1	5434.13	65.78	5501.94
R208	701.2*						
R209	854.8	15609	97	5	1073.17	137.10	1220.33
R210	900.5	16395	126	9	16454.83	280.83	16783.85
R211	746.7*						
Total		228195	691	61	29881.19	1258.28	31281.16

* The instance was not solved to optimality due to memory overflow. This value corresponds to the best incumbent solution that was found by the IPBPC.

Table 5.3: IPBPC results for the 200-series Solomon's instances.

cerning the results for the 200-series instances, the total number of nodes in the IPBPC was a bit larger than the number of nodes in the simplex-based approach in this case. On the other hand, the number of valid inequalities is still smaller for every instance and there is an even larger difference when compared to the number of valid inequalities generated by the simplex-based BPC. For the instance RC203, for example, the simplex-based method generated almost 8 times more valid inequalities than the IPBPC. Large differences are observed for other instances as well, such as RC202 and RC206, which reveals an important advantage of the IPBPC. As discussed in Section 5.2.2, keeping the number of valid inequalities small is crucial for the efficiency of the subproblem solver, as the SR inequalities affect the structure of the subproblem. Considering the total CPU time, the IPBPC outperforms the simplex-based BPC. These results indicate that the use of the primal-dual interior point algorithm within the branch-price-and-cut method offers a significant advantage over the simplex-based approach.

5.4.2 Impact of changes in the core components

To verify the importance of the modifications which we have made on the core components of the IPBPC, we have run some additional computational experiments with different choices of parameters. With these experiments we do not aim to have an exhaustive testing of all possible parameter choices, but rather we want to investigate the impact of changing some key parameters of the method to get a better understanding of the proposed strategy.

	Simplex-based BPC			Ratio		
	Cuts	**Nodes**	**Total time**	**Cuts**	**Nodes**	**Total time**
C101	0	1	2	1.00	1.00	1.56
C102	0	1	8	1.00	1.00	3.63
C103	0	1	28	1.00	1.00	7.04
C104	0	1	86	1.00	1.00	9.34
C105	0	1	3	1.00	1.00	2.03
C106	0	1	4	1.00	1.00	2.13
C107	0	1	4	1.00	1.00	2.27
C108	0	1	7	1.00	1.00	2.47
C109	0	1	16	1.00	1.00	5.06
RC101	87	1	19	2.49	1.00	5.88
RC102	193	3	120	2.30	3.00	8.60
RC103	262	5	541	2.02	1.67	5.73
RC104	437	21	11773	2.04	2.33	5.14
RC105	79	1	33	2.47	1.00	7.49
RC106	755	71	3916	1.40	1.16	4.58
RC107	158	1	161	2.63	1.00	6.36
RC108	228	1	635	2.35	1.00	3.48
R101	19	15	8	4.75	5.00	4.74
R102	0	1	3	1.00	1.00	1.72
R103	53	1	20	2.94	1.00	3.24
R104	391	11	3103	2.13	2.20	7.54
R105	144	3	36	3.35	1.00	5.77
R106	144	3	87	2.77	1.00	6.61
R107	227	4	416	1.79	1.33	5.33
R108	296	1	891	1.79	1.00	2.14
R109	588	65	1127	1.73	1.86	3.24
R110	219	5	426	1.81	1.67	6.87
R111	736	111	5738	1.10	1.04	3.37
R112	574	19	16073	1.38	1.12	10.21
Total	5590	352	45284	1.68	1.31	5.58

Table 5.4: Comparison to a simplex-based BPC method (100-series Solomon's instances).

In the first experiment, we analyze how the early branching strategy described in Sections 5.2.3 and 5.3.4 affects the performance of the IPBPC. In Fig. 5.1 we plot the results of solving all the 100-series instances with different choices of the parameter ε_b, the threshold tolerance for finishing the preprocessing step in the two-step branching approach. Four different values of ε_b were tested: 10^{-3}, 10^{-4}, 10^{-5} and 10^{-6}. Three plots are given in Fig. 5.1 and they show (a) the total number of valid inequalities, (b) the total number of nodes in the search tree, and (c) the total CPU time, for each choice of ε_b. From (a), we see that the values 10^{-3} and 10^{-4} resulted in the smallest numbers of valid inequalities. From (b) and (c), we deduce that $\varepsilon_b = 10^{-3}$ resulted in the smallest number of nodes and the best total CPU time. The use of $\varepsilon_b = 10^{-6}$ in the branching strategy causes a loss of efficiency measured in nearly doubling the CPU time when compared with using $\varepsilon_b = 10^{-3}$. Hence, we conclude that the early branching contributed to the overall efficiency of the IPBPC. Notice that a reduction in the CPU time was expected for larger values of ε_b, as the column and cut generation is stopped earlier and, hence, less calls to the oracle are made. However, the reduction in the number of nodes is an interesting result and it suggests that, indeed, interior point methods have the advantage of quickly approaching the optimal solution. As a consequence of this feature, the provided suboptimal solutions are accurate enough to be safely used in the branching procedure.

The second experiment involves the impact of the parameter ε_c which is used in the oracle of the column and cut generation procedure described in Section 5.2.2. Recall that this parameter has the following purpose: the separation subproblem is called by the oracle

	Simplex-based BPC			Ratio		
	Cuts	**Nodes**	**Total time**	**Cuts**	**Nodes**	**Total time**
C201	0	1	9	1.00	1.00	1.28
C202	0	1	49	1.00	1.00	2.82
C203	0	1	122	1.00	1.00	3.00
C204	0	1	16416	1.00	1.00	22.14
C205	0	1	15	1.00	1.00	2.02
C206	0	1	24	1.00	1.00	1.82
C207	0	1	84	1.00	1.00	4.02
C208	0	1	26	1.00	1.00	1.49
RC201	55	3	92	3.24	1.00	2.18
RC202	39	1	89	4.33	1.00	1.67
RC203	47	1	324	7.83	1.00	0.95
RC205	32	1	111	3.56	1.00	3.03
RC206	73	1	344	8.11	1.00	4.08
RC207	210	5	91405	1.81	1.67	33.96
R201	52	1	78	2.60	1.00	2.15
R202	152	17	1663	2.45	1.89	11.30
R203	78	1	641	4.33	1.00	1.87
R205	345	9	6904	3.08	0.69	7.13
R206	171	1	60608	2.11	0.33	27.98
R207	24	1	11228	2.67	1.00	2.04
R209	248	3	22514	2.56	0.60	18.45
R210	266	5	400904	2.11	0.56	23.89
Total	1792	58	613650	2.59	0.97	19.62

Table 5.5: Comparison to a simplex-based BPC method (200-series Solomon's instances).

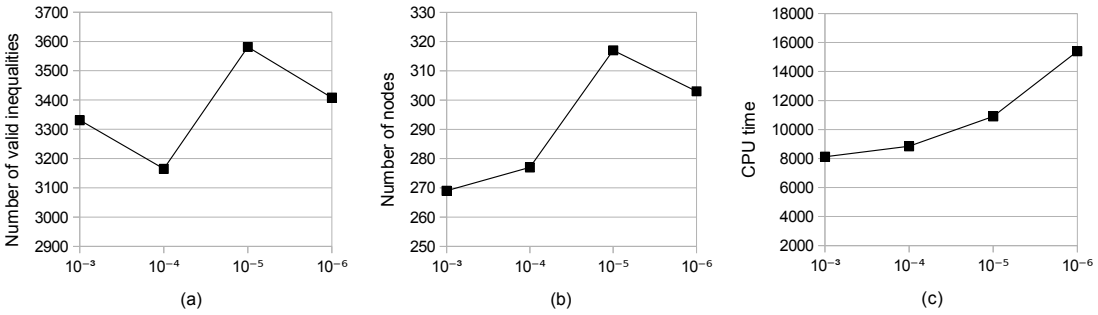

Figure 5.1: Impact of changing the early branching threshold ε_b.

only after the relative gap falls bellow ε_c. Three different values of ε_c were tested in this experiment, namely 0.1, 0.01 and 0.001. In addition, recall that after calling the separation subproblem, up to K_v valid inequalities are added to the RMP. The default value $K_v = 3$ is used in the IPBPC, but for a small ε_c it may be better to allow a larger number of valid inequalities to be added to the RMP. Hence, we have run tests with $K_v = 10$ as well. Fig. 5.2 summarizes the results that were obtained in this experiment. It has three plots with the same meaning as those given in Fig. 5.1. Each plot shows the results for different choices of ε_c; the continuous line corresponds to $K_v = 3$ while the dashed line corresponds to $K_v = 10$. The analysis of these results suggests that the best approach corresponds to calling the separation subproblem as soon as the relative gap in the column and cut generation procedure falls below 0.1. For this scenario, $K_v = 3$ seems to be a more appropriate choice than $K_v = 10$, as the latter resulted in a larger number of generated valid inequalities and larger number of nodes. In addition, the CPU time was considerably increased. The results for $K_v = 10$ were improved when ε_c was reduced, but still they were inferior to the overall performance with $K_v = 3$.

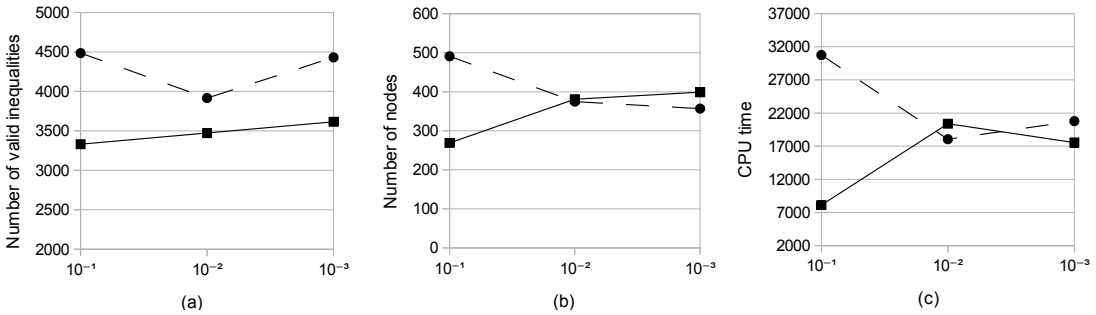

Figure 5.2: Impact of changing the separation subproblem threshold ε_c.

5.5 Concluding remarks

In this chapter, we have addressed the use of the primal-dual interior point algorithm within the branch-price-and-cut method. For each core component of the method, namely the column generation procedure, the separation of valid inequalities (cuts) and the branching procedure, we have presented how to exploit certain advantages that are provided by the interior point algorithm. Two of them are particularly attractive in this context: (*i*) the ability to work with well-centered solutions stabilizes the column and cut generation procedure; (*ii*) the use of early termination improves the overall efficiency of the approach. To verify the behavior of the proposed interior point branch-price-and-cut method, we have run computational experiments on well-known instances of the vehicle routing problem with time windows, a classical integer programming problem. The results provide evidence that the proposed method is efficient and outperforms the state-of-the-art standard branch-price-and-cut method which uses the simplex method to solve the linear relaxations. We have observed considerable reductions in the number of nodes in the search tree, in the number of generated valid inequalities and in the CPU time to solve the instances.

The next steps regarding this investigation will involve solving different integer programming problems by the interior point branch-price-and-cut method proposed here. The focus will be on problems with a large-scale master problem formulation, so that a larger percentage of CPU time is spent on solving the RMPs. Also, we intend to improve our ESPPRC implementation by following the main ideas discussed by Baldacci et al. (2012) in order to solve the Solomon's instances that could not be solved by the current implementation.

Chapter 6

Conclusion

The linear optimization methodologies have been around for more than 50 years. Even though, the research regarding these methodologies is still very active nowadays. In this thesis, we have presented theoretical and computational developments with the aim of contributing with the state-of-the-art in linear optimization. At the end of each chapter, we have presented the concluding remarks regarding the addressed subjects. We now summarize the main outcomes of each chapter.

In Chapter 2, we introduced a unified framework to describe simplex type methods and interior point method in a uniform way. The description presents these methodologies by using the same notation and nomenclature. Hence, it allows to pointing out the main similarities and differences regarding these methods. We believe that this unified framework may be helpful to the newcomers in the area, who want to learn about the main linear programming methodologies. Also, it may be useful to those who are already familiar with simplex type methods, as this framework states these methods in a new perspective.

The dual simplex method for problems in the general form, which we address in Chapter 3, is a competitive variant of simplex type methods. We presented a novel theoretical description for this method, which follows the unified framework that we propose in Chapter 2. In addition, we discussed the main computational techniques which are important to obtain an efficient implementation of the method. We have run computational experiments with the Netlib instances, which are commonly used to benchmark linear programming solvers. The results indicate that dual simplex method for problems in the general form is more efficient than the primal simplex method for problem in the standard form. This variant was around 10% faster than the dual simplex method for problems in the standard form. Therefore, this variant seems promising and should be further investigated. In particular, phase-I strategies may be used to improve the performance of the method. Also, different pricing rules should be studied, including the extension of those which are used in the standard dual simplex method.

The use of the primal-dual interior point algorithm within the column generation method has shown to be a successful strategy. As presented in Chapter 4, the key idea of the primal-dual column generation method is to use the interior point method to obtain suboptimal solutions which are well-centered in the feasible set. Hence, these solutions contribute with the stability of the column generation technique. The results of extensive computational experiments indicate that the primal-dual column generation has the best overall performance in relation to the standard column generation method and the analytic center cutting plane method. The experiments were based on linear relaxations of classical integer programming problems, namely the cutting stock problem, the vehicle routing problem with time windows and the capacitated lot sizing problem with setup times. These problems are described in Appendix A, in which we present the main formulations and how to decompose them by using

the Dantzig-Wolfe decomposition. As a future study, we plan to further verify the performance of this variant in different types of problems, specially those with large-scale formulations. Moreover, it would be interesting to extensively compare the primal-dual column generation method to other stabilized variants such as the bundle method and the volume algorithm.

The advantageous features that are provided by interior point algorithms are also beneficial to the branch-price-and-cut method. In Chapter 5, we show how to combine these two methodologies in order to improve the performance of solving integer programming problems. The proposed interior point branch-price-and-cut method exploits early termination and well-centered primal and dual solutions at the core components of the branch-price-and-cut method, namely the generation of columns, generation of valid inequalities, and branching. We discussed in detail how to modify each of these components efficiently. To verify the performance of the proposed integration, we have presented computational experiments which were based on widely used instances of the vehicle routing problem. The results show that the interior point branch-price-and-cut method is more stable than the standard branch-price-and-cut method on these instances. On average, the number of valid inequalities is considerably smaller in the interior point approach. The CPU time is also smaller than in the standard approach, and the proposed method is on average 20 times faster in the most difficult class of instances. It is worth mentioning that this research was motivated by the low usage of interior point methods within integer programming methodologies. In fact, the vast majority of implementations of the branch-and-price method are based on (extreme) optimal solutions which are obtained by simplex type methods. Therefore, one of the contributions of this investigation is to show that interior point methods can and should be used in the context of integer programming. Further studies on this topic will involve the solution of different classes of problems. Also, we plan to integrate the interior point algorithm with other integer programming methodologies.

In summary, we believe that we have achieved the main goal of this doctoral research, which was to propose new strategies to improve linear optimization methodologies. This research involved the study of different types of theoretical subjects and computational methods, which are usually treated separately in the literature. Indeed, not many publications deal with both simplex type methods and interior point methods. Moreover, few papers deal with the use of interior point methods within integer programming methodologies. In this study, we have put together different subjects of the linear optimization field with the purpose of exploiting the strength of each one and, hence, to improve the state-of-the-art methodologies.

Appendix A

The Dantzig-Wolfe decomposition

In this Appendix, we describe the fundamental concepts of the Dantzig-Wolfe decomposition (DWD) and how to apply it to integer programming problems. The DWD is a technique proposed to decompose a linear programming problem which has a special structure in the coefficient matrix. The original aim of this technique consisted in making large linear problems tractable as well as to speed up the solution by the simplex method (Dantzig and Wolfe, 1960). Except for some classes of problems, the DWD was not advantageous for general linear programming problems. However, it showed to be very successful when extended to integer programming problems (Barnhart et al., 1998; Vanderbeck, 2000; Lübbecke and Desrosiers, 2005). In this context, the focus is to provide stronger bounds when solving linear relaxations in order to speed up a branch-and-bound search.

The purpose of this Appendix is to support the developments presented in Chapters 4 and 5, as the formulations that are addressed in those chapters are obtained by using the DWD. Moreover, the description presented here may also be useful to the reader that is interested in an introduction to the DWD. As complementary readings, we suggest the excellent papers by Lübbecke and Desrosiers (2005) and Vanderbeck and Wolsey (2010). The remainder sections in this Appendix are the following. In Section A.1, we describe the DWD for integer programming problems. This decomposition is equivalent to the Lagrangian relaxation, as showed in Section A.2. Finally, we illustrate the use of the DWD on three classical integer programming problems: the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST).

A.1 DWD for integer programming problems

The DWD is applied to formulations with the following special structure. The coefficient matrix is very sparse and composed by isolated *blocks* of non-zeros. In addition, the matrix has a set of rows given by *linking constraints*, which links most of the variables related to the blocks of non-zeros. By discarding these constraints, the blocks would be independent from each other. Fig. A.1 illustrates a coefficient matrix having the described features.

Consider the following integer programming problem, which will be referred to as the *compact formulation*:

$$\min \quad c^T x, \tag{A.1a}$$

$$\text{s.t.} \quad Ax = b, \tag{A.1b}$$

$$x \in \mathcal{X}, \tag{A.1c}$$

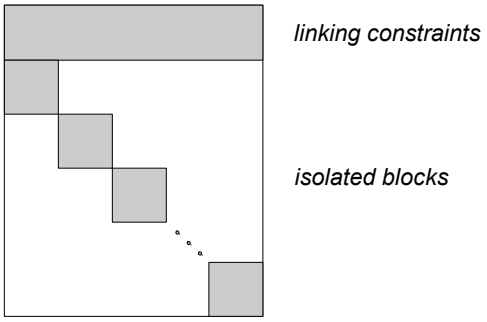

Figure A.1: The coefficient matrix that has the special structure which is suitable for applying the DWD.

where $\mathcal{X} = \{x \in \mathbb{Z}^n_+ : Dx = d\}$ is a discrete set, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $D \in \mathbb{R}^{h \times n}$, $d \in \mathbb{R}^h$. The constraints (A.1b) are the linking constraints in the formulation. We assume the problem is easier to solve by taking advantage of the structure of $\mathcal{X}$, in particular of the matrix D. Hence, by applying the DWD we are likely to obtain a more advantageous formulation.

To apply the DWD to (A.1), we consider the convexification approach, although alternative approaches can be used as well (see e.g. Vanderbeck and Wolsey, 2010). Let $\mathcal{C} = conv(\mathcal{X})$ denote the convex hull of the set $\mathcal{X}$. The convex hull is the tightest polyhedron which contains all the integer points in $\mathcal{X}$. As a polyhedron, it can be fully represented by all its extreme points p_q and extreme rays p_r, as stated in Theorem A.1.1.

Theorem A.1.1 (Resolution theorem)**.** *Let $X = \{x \in \mathbb{R} \mid Ax \geq b\}$ be a nonempty polyhedron with at least one extreme point. Let $[p_q]_{q \in Q}$ be the extreme points, and let $[p_r]_{r \in R}$ be a complete set of extreme rays of X, where Q and R are the respective sets of indices. Let*

$$C = \left\{ \sum_{q \in \mathcal{Q}} \lambda_q p_q + \sum_{r \in \mathcal{R}} \mu_r p_r | \sum_{q \in \mathcal{Q}} \lambda_q = 1, \lambda_q \geq 0, \mu_r \geq 0 \right\}.$$

Then $C = X$.

Theorem A.1.1 is also known as the *Representation Theorem* and the *Caratheodory's Theorem*. Its proof can be found in several text books (see e.g. Bertsimas and Tsitsiklis, 1997; Bazaraa et al., 1990). Notice that even though we are considering equality constraints in formulation (A.1), the theorem is still valid for this case, as each equality constraint can be equivalently rewritten as two inequalities.

As a consequence of Theorem A.1.1, any $x \in \mathcal{C}$ can be rewritten as a convex combination of the extreme points and rays of $\mathcal{C}$, *i.e.*,

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

By using this correspondence in problem (A.1), we obtain the equivalent formulation:

$$\min \quad \sum_{q \in \mathcal{Q}} \lambda_q (c^T p_q) + \sum_{r \in \mathcal{R}} \mu_r (c^T p_r), \tag{A.2a}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{Q}} \lambda_q (A p_q) + \sum_{r \in \mathcal{R}} \mu_r (A p_r) = b, \tag{A.2b}$$

$$\sum_{q\in\mathcal{Q}} \lambda_q = 1, \tag{A.2c}$$

$$\lambda_q \geq 0,\ \mu_r \geq 0, \qquad \forall q \in \mathcal{Q}, \forall r \in \mathcal{R}, \tag{A.2d}$$

$$x = \sum_{q\in\mathcal{Q}} \lambda_q p_q + \sum_{r\in\mathcal{R}} \mu_r p_r, \tag{A.2e}$$

$$x \in \mathbb{Z}^n_+. \tag{A.2f}$$

Notice that we still need to keep $x \in \mathbb{Z}^n_+$ in order to guarantee the equivalence between (A.2) and (A.1). The linear relaxation of (A.2) usually leads to a lower bound that is the same as or stronger than the one obtained by the linear relaxation of (A.1). For this reason, formulation (A.2) is more advantageous than (A.1) when using a branch-and-bound approach to solve the problem.

Let us focus now on the linear relaxation of (A.2). Since constraints (A.2f) are dropped, there is no need to keep constraints (A.2e). By denoting $c_j = c^T p_j$ and $a_j = Ap_j$, $\forall j \in \mathcal{Q}$ and $\forall j \in \mathcal{R}$, a *relaxation* of the problem (A.2) is given by:

$$\min \quad \sum_{q\in\mathcal{Q}} c_q\lambda_q + \sum_{r\in\mathcal{R}} c_r\mu_r \tag{A.3a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}} a_q\lambda_q + \sum_{r\in\mathcal{R}} a_r\mu_r = b, \tag{A.3b}$$

$$\sum_{q\in\mathcal{Q}} \lambda_q = 1, \tag{A.3c}$$

$$\lambda_q \geq 0,\ \mu_r \geq 0, \qquad \forall q \in \mathcal{Q}, \forall r \in \mathcal{R}, \tag{A.3d}$$

which is called Dantzig-Wolfe *master problem*. The columns of the coefficient matrix of this formulation, are given by linear transformations of all extreme points and extreme rays of $\mathcal{C}$. Therefore, the number of variables in the master problem is huge, typically exponential. For this reason, the column generation method is commonly used to solve this formulation (see Chapter 4). Indeed, the columns of the master problem can be generated by solving the *subproblem*

$$\begin{aligned} z_{SP}(u,v) &:= \min_{q\in\mathcal{Q}, r\in\mathcal{R}} \left\{0, c_q - u^T a_q - v, c_r - u^T a_r\right\}, \\ &= \min_{x\in\mathcal{C}} \left\{0, (c - A^T u)^T x - v\right\}, \end{aligned}$$

where $u \in \mathbb{R}^m$ and $v \in \mathbb{R}$ denote the dual variables associated to constraints (A.3b) and (A.3c), respectively.

Recall we have assumed that the matrix D in set $\mathcal{X}$ has a special block structure. It allows the matrix to be partitioned in several independent submatrices D^k, $k = 1, \ldots, K$. As a result, the set $\mathcal{X}$ is represented as the Cartesian product of K independent sets. Let us define $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_K$, where

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

where $|L_k|$ is the number of variables associated to $\mathcal{X}_k$, and x^k is the vector containing the components of x associated to $\mathcal{X}_k$. For simplicity, we assume the set $\mathcal{X}$ is bounded and, hence, $\mathcal{R} = \varnothing$, although the following discussion can be extended to deal with unbounded cases (see e.g. Vanderbeck and Wolsey, 2010). Following the same ideas as described so far, problem (A.3) can be rewritten as

$$\min \quad \sum_{k=1}^{K} \sum_{q\in\mathcal{Q}_k} c^k_q \lambda^k_q \tag{A.4a}$$

$$\text{s.t.} \quad \sum_{k=1}^{K} \sum_{q \in \mathcal{Q}_k} a_q^k \lambda_q^k = b, \tag{A.4b}$$

$$\sum_{q \in \mathcal{Q}_k} \lambda_q^k = 1, \qquad \forall k = 1, \ldots, K, \tag{A.4c}$$

$$\lambda_q^k \geq 0, \qquad \forall q \in \mathcal{Q}_k,\ \forall k = 1, \ldots, K, \tag{A.4d}$$

where the extreme points of the subset $\mathcal{X}_k$ are represented by each p_q with $q \in \mathcal{Q}_k$. Hence, the columns in this formulation are given by the extreme points from K subsets. When using the column generation method to solve (A.4), up to K columns can be generated at each outer iteration, one from each subproblem

$$z_{SP}^k(u, v_k) := \min \left\{ (c^k - (A^k)^T u)^T x^k - v_k \mid x^k \in \mathcal{X}_k \right\}, \quad k = 1, \ldots, K,$$

where A^k are the columns in A associated to the variables x^k, and $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^K$ denote the dual variables associated to constraints (A.4b) and (A.4c), respectively.

In certain problems, the K subsets $\mathcal{X}_k$ are identical and hence they lead to the same extreme points and rays. Therefore, one of them is enough to fully represent all the columns in (A.4), and the master variables can be aggregated as

$$\lambda_q := \sum_{k=1}^{K} \lambda_q^k. \tag{A.5}$$

As a consequence, we can drop the index k from Q_k and denote it simply by Q, since all Q_k represent the same set. The same simplification may be applied to the parameters c_q^k and a_q^k. Considering all these changes together, we can rewrite problem (A.4) as the following *aggregated* master problem:

$$\min \quad \sum_{q \in \mathcal{Q}} c_q \lambda_q \tag{A.6a}$$

$$\text{s.t.} \quad \sum_{q \in \mathcal{Q}} a_q \lambda_q = b, \tag{A.6b}$$

$$\sum_{q \in \mathcal{Q}} \lambda_q = K, \tag{A.6c}$$

$$\lambda_q \geq 0, \qquad \forall q \in \mathcal{Q}. \tag{A.6d}$$

The aggregation reduces the dimensions of the master problem and only one of the subproblems is used in a column generation method. If $0 \in \mathcal{X}_k$ and its associated cost is zero, then the equality in constraint (A.6c) can be relaxed to

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

In addition, if K is sufficiently large, then this inequality holds strictly in the optimal solution and, hence, (A.6c) can be dropped from the problem.

A.2 Equivalence to Lagrangian relaxation

The dual problem associated to (A.3) has the same form as the problem we obtain by applying Lagrangian relaxation for integer programming to the compact formulation (A.1) (see

Geoffrion (1974)). To see this, we associate a vector of Lagrange multipliers u to constraints (A.1b), and use them to penalize the violation of these constraints. Recall that $\mathcal{C}$ denotes the convex hull of $\mathcal{X}$. We define the Lagrangian subproblem as

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

For an arbitrary $\overline{u}$, we obtain a lower bound for the optimal value of problem (A.1) by solving $L_D(\overline{u})$. The best lower bound we can obtain is given by the Lagrangian dual problem

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

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

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

which is the dual problem of (A.3). It shows the relationship between DWD and Lagrangian relaxation. Furthermore, to solve this linear programming problem we typically use the Kelley's cutting plane method (Kelley, 1960), in which we start with subsets $\mathcal{Q}' \in \mathcal{Q}$ and $\mathcal{R}' \in \mathcal{R}$, and then generate other constraints iteratively, by recurring to the Lagrangian subproblem $L_D(u)$. This row generation in the dual space is equivalent to the column generation in the primal space.

A.3 Examples of applying the DWD

In this section, we describe how to decompose three classical integer programming problems by using the DWD. The problems we address are the following: the cutting stock problem (CSP), the vehicle routing problem with time windows (VRPTW), and the capacitated lot sizing problem with setup times (CLSPST). Different types of master problem formulations are obtained for these problems: an aggregated master problem in the CSP, an aggregated master problem with a set partitioning structure in the VRPTW, and a disaggregated master problem in the CLSPST. We also discuss for each problem how the column generation method can be used for solving the resulting master problem formulation.

A.3.1 Cutting stock problem

The one-dimensional CSP consists in determining the smallest number of stock rolls of width W that have to be cut in order to satisfy the demands d_j of pieces of width w_j, $j \in M = \{1, 2, \ldots, m\}$. We assume there is an upper bound n on the number of stock rolls needed to satisfy the demands and, hence, we associate to each roll an index in $N = \{1, 2, \ldots, n\}$. A compact formulation for the problem is given by

$$\min \quad \sum_{i \in N} y_i, \tag{A.8a}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ij} \geq d_j \qquad \forall j \in M, \tag{A.8b}$$

$$\sum_{j \in M} w_j x_{ij} \leq W y_i \qquad \forall i \in N, \tag{A.8c}$$

$$y_i \in \{0,1\}, \qquad \forall i \in N, \tag{A.8d}$$

$$x_{ij} \geq 0 \text{ and integer}, \qquad \forall i \in N, \forall j \in M, \tag{A.8e}$$

where $y_i = 1$ if the roll i is used, and 0 otherwise. The number of times a piece of width w_j is cut from roll i is denoted by x_{ij}. Constraints (A.8b) guarantee that all demands must be satisfied, and constraints (A.8c) enforce that the sum of the widths of all pieces cut from a roll does not exceed its width W.

The coefficient matrix of problem (A.8) has a special structure which is well-suited to the application of the DWD. The linking constraints in this formulation are given by (A.8b). Consider the set $\mathcal{X}$ of all points that satisfy constraints (A.8c), (A.8d) and (A.8e). Following the discussion presented in Section A.1, we define the subsets $\mathcal{X}_i$, for each $i \in N$, which are independent to each other and satisfy $\mathcal{X}_1 \times \ldots \times \mathcal{X}_n = \mathcal{X}$. Furthermore, we replace each $\mathcal{X}_i$ by its convex hull $conv(\mathcal{X}_i)$, which is a bounded set and hence its set of extreme points is enough to fully describe this set. For each $i \in N$, let P_i be the set of indices of all extreme points of $conv(\mathcal{X}_i)$. These extreme points are then denoted by $(y_p^i, x_{p1}^i, \ldots, x_{pm}^i)$, for each $p \in P_i$. Following this notation, we have the master problem:

$$\min \quad \sum_{i \in N} \sum_{p \in P_i} y_p^i \lambda_p^i, \tag{A.9a}$$

$$\text{s.t.} \quad \sum_{i \in N} \sum_{p \in P_i} x_{pj}^i \lambda_p^i \geq d_j, \qquad \forall j \in M, \tag{A.9b}$$

$$\sum_{p \in P_i} \lambda_p^i = 1, \qquad \forall i \in N, \tag{A.9c}$$

$$\lambda_p^i \geq 0, \qquad \forall i \in N, \forall p \in P_i. \tag{A.9d}$$

Since we have assumed the stock pieces are identical, the subsets $\mathcal{X}_i$ are the same for every $i \in N$ and hence they will lead to the same columns (regarding the coefficients in the linking constraints). This situation should be avoided by aggregating the master variables as in (A.5). The resulting aggregated master problem is

$$\min \quad \sum_{p \in P} y_p \lambda_p, \tag{A.10a}$$

$$\text{s.t.} \quad \sum_{p \in P} x_{pj} \lambda_p \geq d_j, \qquad \forall j \in M, \tag{A.10b}$$

$$\lambda_p \geq 0, \qquad \forall p \in P, \tag{A.10c}$$

where P represents the set of indices of all extreme points of $conv(\bar{\mathcal{X}})$, with $\bar{\mathcal{X}} := \mathcal{X}_1 = \ldots = \mathcal{X}_n$. Also, we dropped the convexity constraint, as $\mathbf{0} \in conv(\bar{\mathcal{X}})$ and n is an inactive upper bound in the constraint $\sum_{p \in P} \lambda_p \leq n$.

Due to the large number of columns in formulation (A.9), it is typically solved by the column generation method. Let $u \in \mathbb{R}^m$ be the vector of dual variables associated to constraints (A.10b). The associated oracle is given by a subproblem of the form

$$\min \quad y_i - \sum_{j \in M} \overline{u}_j x_j, \tag{A.11a}$$

$$\text{s.t.} \quad (y_i, x_1, \ldots, x_m) \in conv(\bar{\mathcal{X}}), \tag{A.11b}$$

where $\overline{u}$ represents an arbitrary dual solution. To obtain the solution of this subproblem, we first solve a knapsack problem given by

$$\max \quad \sum_{j \in M} \overline{u}_j x_j, \tag{A.12a}$$

$$\text{s.t.} \quad \sum_{j \in M} w_j x_j \leq W, \tag{A.12b}$$

$$x_j \geq 0 \text{ and integer}, \quad \forall j \in M. \tag{A.12c}$$

Then, an optimal solution $(x_1^\star, \ldots, x_m^\star)$ of (A.12) is used to generate a column of (A.10) as follows. If $1 - \sum_{j \in M} \overline{u}_j x_j^\star < 0$, then the column is generated by setting $y_p := 1$ and $x_{pj} := x_j^\star$ for all $j \in M$. Otherwise, we assume the solution is given by an empty pattern and, hence, the column is generated by setting $y_p := 0$ and $x_{pj} := 0$ for all $j \in M$. If the k-best solutions of the knapsack problem are available, for a given $k > 0$, then up to k columns can be generated at each call to the oracle.

A.3.2 Vehicle routing problem with time windows

Consider a fleet of vehicles $V = \{1, 2, \ldots, v\}$ available to service a set of customers $C = \{1, 2, \ldots, n\}$ with demands d_i, $i \in C$. We assume all the vehicles are identical and are initially at a same depot, and every route must start and finish at this depot. A vehicle can serve more than one customer in a route, as long as its maximum capacity q is not exceeded. Each customer $i \in C$ must be visited only once and the visiting time must satisfy a time window $[w_i^a, w_i^b]$. Late arrivals (after time w_i^b) are not allowed and if a vehicle arrives earlier than w_i^a, then it needs to wait until the window opens. In addition, a service time s_i is assigned to each customer $i \in C$. The objective in the problem is to design a set of minimum cost routes so that the described requirements are satisfied.

Let $N = \{0, 1, \ldots, n, n+1\}$ be a set of vertices such that vertices 0 and $n+1$ represent the depot, and the remaining vertices correspond to the customers in C. The time of travelling from vertex i to vertex j, denoted by t_{ij}, satisfies the triangle inequality. By using this notation, we can formulate the VRPTW as follows:

$$\min \quad \sum_{k \in V} \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ijk} \tag{A.13a}$$

$$\text{s.t.} \quad \sum_{k \in V} \sum_{j \in N} x_{ijk} = 1, \qquad \forall i \in C, \tag{A.13b}$$

$$\sum_{i \in C} d_i \sum_{j \in N} x_{ijk} \leq q, \qquad \forall k \in V, \tag{A.13c}$$

$$\sum_{j \in N} x_{0jk} = 1, \qquad \forall k \in V, \tag{A.13d}$$

$$\sum_{i \in N} x_{ihk} - \sum_{j \in N} x_{hjk} = 0, \qquad \forall h \in C,\ \forall k \in V, \tag{A.13e}$$

$$\sum_{i \in N} x_{i,n+1,k} = 1, \qquad \forall k \in V, \tag{A.13f}$$

$$w_{ik} + t_{ij} + s_i - M_{ij}(1 - x_{ijk}) \leq w_{jk}, \qquad \forall\ i, j \in N,\ \forall k \in V, \tag{A.13g}$$

$$w_i^a \leq w_{ik} \leq w_i^b, \qquad \forall i \in N,\ \forall k \in V, \tag{A.13h}$$

$$x_{ijk} \in \{0,1\}, \qquad \forall\ i,j \in N,\ \forall k \in V. \tag{A.13i}$$

The binary variable x_{ijk} determines whether vehicle $k \in V$ visits vertex $i \in N$ and then goes immediately to vertex $j \in N$. The variable w_{ik} determines the time that vehicle $k \in V$ arrives at customer i. We assume that all the parameters are non-negative integers, c_{ij} is given by the Euclidean distance between vertices i and j, and M_{ij} is a sufficiently large number (*e.g.*, $M_{ij} = \max\{0, w_i^b + t_{ij} + s_i - w_j^a\}$). Constraints (A.13b) guarantee that each customer must be visited by only one vehicle. Constraints (A.13c) enforce that a vehicle cannot exceed its capacity. Both constraints together ensure that the demand of each client has to be satisfied by only one vehicle. Moreover, constraints (A.13d) and (A.13f) enforce that each vehicle must start and finish its route at the depot, respectively. Constraints (A.13e) guarantee that once a vehicle visits a customer and serves it, it must then move to another customer or end its route at the depot (vertex $n+1$). Constraints (A.13g) establish the relationship between the vehicle departure time from a customer and its immediate successor. Indeed, if $x_{ijk} = 1$ then the constraint becomes $w_{ik} + t_{ij} \leq w_{jk}$. Constraints (A.13h) enforce that the vehicle k serves customer i inside the time window $[w_i^a, w_i^b]$. Note that if a vehicle is not used, its route is defined as $(0, n+1)$.

The coefficient matrix of formulation (A.13) has a special structure that can be exploited by the DWD, taking (A.13b) as the linking constraints. Let $\mathcal{X}$ be the set of all points satisfying constraints (A.13c) to (A.13i). We can define v independent subsets $\mathcal{X}_k$ from $\mathcal{X}$, for each $k \in V$, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_v$. We replace each $\mathcal{X}_k$ by its convex hull $conv(\mathcal{X}_k)$, which is a bounded set and hence is fully represented by its set of extreme points. For each $k \in V$, P_k represents the set of indices of all extreme points of $conv(\mathcal{X}_k)$. Following developments in Section A.1, we obtain the master problem

$$\min \quad \sum_{k\in V}\sum_{i\in N}\sum_{j\in N}\sum_{p\in P_k} c_{ij}x_{ijp}^k\lambda_p^k \tag{A.14a}$$

$$\text{s.t.} \quad \sum_{k\in V}\sum_{j\in N}\sum_{p\in P_k} x_{ijp}^k\lambda_p^k = 1, \qquad \forall i \in C, \tag{A.14b}$$

$$\sum_{p\in P_k} \lambda_p^k = 1, \qquad \forall k \in V, \tag{A.14c}$$

$$\lambda_p^k \geq 0, \qquad \forall k \in V, \forall p \in P_k, \tag{A.14d}$$

where for a given $p \in P_k$, x_{ijp}^k are the components of the corresponding extreme point of $conv(\mathcal{X}_k)$, for all $i,j \in N$. We assume that all the vehicles are identical, so the subsets $\mathcal{X}_k$ will be the same for every $k \in V$. This characteristic allows us to aggregate the master variables by using the redefinition given in (A.5). As a result, we obtain the aggregated master problem

$$\min \quad \sum_{i\in N}\sum_{j\in N}\sum_{p\in P} c_{ij}x_{ijp}\lambda_p \tag{A.15a}$$

$$\text{s.t.} \quad \sum_{j\in N}\sum_{p\in P} x_{ijp}\lambda_p = 1, \qquad \forall i \in C, \tag{A.15b}$$

$$\lambda_p \geq 0, \qquad \forall p \in P, \tag{A.15c}$$

where P is the set of indices of all extreme points of $conv(\bar{\mathcal{X}})$, with $\bar{\mathcal{X}} := \mathcal{X}_1 = \ldots = \mathcal{X}_v$. The convexity constraint has been dropped since $\mathbf{0} \in conv(\bar{\mathcal{X}})$ and v is a loose upper bound in the constraint $\sum_{p\in P} \lambda_p \leq v$.

We consider now solving the aggregated formulation (A.15) by the column generation method. Let $u = (u_1, \ldots, u_n)$ denote the vector of dual variables associated to constraints

(A.15b). 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 (A.15) is given by the subproblem

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

This subproblem is an elementary shortest path problem with resource constraints (ESPPRC) – see Section 5.3.2 for a further description regarding this problem. An optimal solution $[x^\star_{ij}, s^\star_i]_{i,j \in N}$ of the ESPPRC is an extreme point of $conv(\bar{\mathcal{X}})$. To generate a column of (A.15), we set $x_{ijp} = x^\star_{ij}$, for all $i, j \in N$. If the k-best solutions of the ESPPRC are available, then we can generate up to k columns in a call to the oracle.

Due to the difficulty in solving the ESPPRC, some implementations ignore the elementarity requirement. In other words, they allow the shortest paths to contain cycles and, hence, a customer may be visited more than once. By using this approach, the CPU time to solve the subproblem is substantially reduced. However, the lower bound provided by the master problem formulation may be slightly worse (see e.g. Irnich and Desaulniers, 2005; Righini and Salani, 2008).

A.3.3 Capacitated Lot-Sizing Problem with Setup Times

Consider a set $M = \{1, \ldots, m\}$ of items that must be processed by a single machine, during a planning horizon given by the set $N = \{1, \ldots, n\}$ of time periods. 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). Consider the following compact formulation proposed by Trigeiro et al. (1989)

$$\min \quad \sum_{t \in N} \sum_{j \in M} (c_{jt} x_{jt} + h_{jt} s_{jt} + f_{jt} y_{jt}) \tag{A.16a}$$

$$\text{s.t.} \quad \sum_{j \in M} (a_{jt} x_{jt} + b_{jt} y_{jt}) \leq C_t, \qquad \forall t \in N \tag{A.16b}$$

$$s_{j(t-1)} + x_{jt} = d_{jt} + s_{jt}, \qquad \forall j \in M, \forall t \in N, \tag{A.16c}$$

$$x_{jt} \leq D y_{jt}, \qquad \forall j \in M, \forall t \in N, \tag{A.16d}$$

$$x_{jt} \geq 0, \qquad \forall j \in M, \ \forall t \in N, \tag{A.16e}$$

$$s_{jt} \geq 0, \qquad \forall j \in M, \ \forall t \in N, \tag{A.16f}$$

$$y_{jt} \in \{0, 1\}, \qquad \forall j \in M, \ \forall t \in N, \tag{A.16g}$$

where x_{jt} represents the production level of item j in time period t and s_{jt} is the number of units in stock of item j at the end of time period t. Also, the final inventory for every product j is set to zero (*i.e.*, $s_{jn} = 0$). The binary variable y_{jt} determines whether item j is produced in time period t ($y_{jt} = 1$) or not ($y_{jt} = 0$). Constraints (A.16b) enforce that the elapsed time in period t for a given plan of production should not exceed the capacity of the machine in that period. Constraints (A.16c) are the inventory equations which ensure that the production and units of each item in stock at the beginning of a given period must satisfy the demand while the remaining units are stored for next time period. Constraints

(A.16d) guarantee that if item j is produced in period t, then the machine must be set up, where D is a sufficiently large number. Constraints (A.16e) and (A.16f) ensure that the level of production and stock at each period t for each item j are non-negative.

We now exploit the special structure of the coefficient matrix in formulation (A.16), by taking (A.16b) as the linking constraints. The set $\mathcal{X}$ consists of all the points satisfying constraints (A.16c) to (A.16g). For each $j \in M$, we define a subset $\mathcal{X}_j$ by fixing j in $\mathcal{X}$, such that $\mathcal{X} = \mathcal{X}_1 \times \ldots \times \mathcal{X}_m$. Following Section A.1, we replace each $\mathcal{X}_j$ by its convex hull $conv(\mathcal{X}_j)$, which is fully represented by its extreme points ($conv(\mathcal{X}_j)$ is a bounded set). After rewriting the variables of the compact formulation (A.16) in terms of the extreme points of $conv(\mathcal{X}_j)$, for each $j \in M$, we obtain the master problem

$$\min \quad \sum_{j \in M} \sum_{t \in N} \sum_{p \in P_j} \left(c_{jt} x^j_{pt} + h_{jt} s^j_{pt} + f_{jt} y^j_{pt} \right) \lambda^j_p \tag{A.17a}$$

$$\text{s.t.} \quad \sum_{j \in M} \sum_{p \in P_j} \left(a_{jt} x^j_{pt} + b_{jt} y^j_{pt} \right) \lambda^j_p \leq C_t, \qquad \forall t \in N, \tag{A.17b}$$

$$\sum_{p \in P_j} \lambda^j_p = 1, \qquad \forall j \in M, \tag{A.17c}$$

$$\lambda^j_p \geq 0, \qquad \forall j \in M, \forall p \in P_j, \tag{A.17d}$$

where P_j is the set of indexes of all extreme points of $conv(\mathcal{X}_j)$, for each $j \in M$. Each $\mathcal{X}_j$ corresponds to a different set in this formulation, so we keep the formulation disaggregated.

Formulation (A.17) is typically solved by the column generation method. Let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_m)$ denote the vectors of dual variables associated to constraints (A.17b) and (A.17c), 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. Since we have a disaggregated formulation, the oracle is given by m subproblems. For each $j \in M$, we have the subproblem

$$\min \quad \sum_{t \in N} [(c_{jt} - a_{jt}\overline{u}_t) x_{jt} + h_{jt} s_{jt} + (f_{jt} - b_{jt}\overline{u}_t) y_{jt}] - \overline{v}_j \tag{A.18a}$$

$$\text{s.t.} \quad [x_{jt}, s_{jt}, y_{jt}]_{t \in N} \in conv(\mathcal{X}_j), \tag{A.18b}$$

which is a single-item lot sizing problem without capacity constraints. Hence, it can be solved by the Wagner-Whitin algorithm (Wagner and Whitin, 1958). For a given $j \in M$, if the optimal value of the subproblem is negative, the corresponding optimal solution $[x^\star_{jt}, s^\star_{jt}, y^\star_{jt}]_{t \in N}$ is used to generate a column of (A.17) by setting $x^j_{pt} = x^\star_{jt}$, $s^j_{pt} = s^\star_{jt}$ and $y^j_{pt} = y^\star_{jt}$. Otherwise, the solution is discarded and no column is generated from that subproblem. Since m subproblems are solved in each call to the oracle, up to m columns can be generated at each outer iteration.

Bibliography

Aho, A. V.; Hopcroft, J. E.; Ullman, J. D. *Data structures and algorithms.* Addison Wesley, 1983.

Aloise, D.; Hansen, P.; Liberti, L. An improved column generation algorithm for minimum sum-of-squares clustering. *Mathematical Programming*, vol. 131, pp. 195–220, 2012.

Arenales, M. N. *Programação linear.* Ph.D. thesis, Faculdade de Engenharia de Campinas - Universidade Estadual de Campinas, 1984.

Arenales, M. N.; Armentano, V. A.; Morabito, R.; Yanasse, H. *Pesquisa operacional.* Editora Campus, 2007.

Atkinson, D. S.; Vaidya, P. M. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, vol. 69, pp. 1–43, 1995.

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

Baldacci, R.; Mingozzi, A.; Roberti, R. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, vol. 218, no. 1, pp. 1–6, 2012.

Barahona, F.; Anbil, R. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, vol. 87, pp. 385–399, 2000.

Barnhart, C.; Johnson, E. L.; Nemhauser, G. L.; Savelsbergh, M. W. P.; Vance, P. H. Branch-and-price: column generation for solving huge integer programs. *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998.

Bazaraa, M. S.; Jarvis, J. J.; Sherali, H. D. *Linear programming and network flows.* 2 edn.. John Wiley & Sons Inc., 1990.

Beasley, J. E.; Christofides, N. An algorithm for the resource constrained shortest path problem. *Networks*, vol. 19, no. 4, pp. 379–394, 1989.

Ben Amor, H. M.; Desrosiers, J.; Frangioni, A. On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematics*, vol. 157, no. 6, pp. 1167 – 1184, 2009.

Benichou, M.; Gauthier, J. M.; Hentges, G.; Ribiere, G. The efficient solution of large-scale linear programming problems - some algorithmic techniques and computational results. *Mathematical programming*, vol. 13, pp. 280–322, 1977.

Benson, H. Y.; Shanno, D. F. An exact primal-dual penalty method approach to warmstarting interior-point methods for linear programming. *Journal Computational Optimization and Applications*, vol. 38, no. 3, pp. 371–399, 2007.

Bertsimas, D.; Tsitsiklis, J. N. *Introduction to Linear Optimization.* Athena Scientific, Belmont, Massachusetts, 1997.

Bixby, R. E. Solving real-world linear programs: A decade and more of progress. *Operations Research*, vol. 50, no. 1, pp. 3–15, 2002.

Bixby, R. E.; Gregory, J. W.; Lustig, I. J.; Marsten, R. E.; Shanno, D. F. Very large-scale linear programming: a case study in combining interior point and simplex methods. *Operations Research*, vol. 40, no. 5, pp. 885–897, 1992.

Borchers, B.; Mitchell, J. E. *Using an interior point method in a branch and bound algorithm for integer programming.* Tech. Rep. 195, Mathematical Sciences, Rensselaer Polytechnic Institute, 1992.

Braysy, O.; Gendreau, M. Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation Science*, vol. 39, no. 1, pp. 104–118, 2005.

Briant, O.; Lemarechal, C.; Meurdesoif, P.; Michel, S.; Perrot, N.; Vanderbeck, F. Comparison of bundle and classical column generation. *Mathematical programming*, vol. 113, no. 2, pp. 299–344, 2008.

Chabrier, A. Vehicle routing problem with elementary shortest path based column generation. *Computers and Operations Research*, vol. 33, no. 10, pp. 2972 – 2990, 2006.

Clarke, G.; Wright, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, vol. 12, no. 4, pp. 568–581, 1964.

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

COIN-OR Foundation, Inc. *Computational infrastructure for operations research (COIN-OR).* 2012.

Colombo, M.; Gondzio, J. Further development of multiple centrality correctors for interior point methods. *Computational Optimization and Applications*, vol. 41, pp. 277–305, 2008.

Cook, W.; Rich, J. L. *A parallel cutting plane algorithm for the vehicle routing problem with time windows.* Tech. Rep. TR99-04, Computational and Applied Mathematics, Rice University, 1999.

Dantzig, G. B. Maximization of a linear function subject to linear inequalities, In: Koopmans, T. C., Activity Analysis of Production and Allocation, pp. 339–347. 1951.

Dantzig, G. B.; Orchard-Hays, W. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, vol. 8, no. 46, pp. 64–67, 1954.

Dantzig, G. B.; Thapa, M. N. *Linear programming*, vol. 1. Springer Series in Operations Research, 1997.

Dantzig, G. B.; Wolfe, P. Decomposition principle for linear programs. *Operations Research*, vol. 8, no. 1, pp. 101–111, 1960.

Degraeve, Z.; Peeters, M. Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *INFORMS Journal on Computing*, vol. 15, no. 1, pp. 58–81, 2003.

Desaulniers, G.; Desrosiers, J.; Spoorendonk, S. *The vehicle routing problem with time windows: State-of-the-art exact solution methods,* Wiley Encyclopedia of Operations Research and Management Science, John Wiley & Sons, Inc., 2010.

Desaulniers, G.; Desrosiers, J.; Spoorendonk, S. Cutting planes for branch-and-price algorithms. *Networks*, 2011.

Desaulniers, G.; Lessard, F.; Hadjar, A. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, vol. 42, no. 3, pp. 387–404, 2008.

Desrochers, M. *An algorithm for the shortest path problem with resource constraints.* Tech. Rep., Technical Report G-88-27, GERAD, 1988.

Desrochers, M.; Desrosiers, J.; Solomon, M. A new optimization algorithm for the vehicle-routing problem with time windows. *Operations Research*, vol. 40, no. 2, pp. 342–354, 1992.

Desrosiers, J.; Lübbecke, M. E. *Branch-price-and-cut algorithms* John Wiley & Sons, Inc., 2010.

Desrosiers, J.; Soumis, F.; Desrochers, M. Routing with time windows by column generation. *Networks*, vol. 14, no. 4, pp. 545–565, 1984.

Dongarra, J.; Sullivan, F. Guest editors' introduction: The top 10 algorithms. *Computing in Science Engineering*, vol. 2, no. 1, pp. 22 –23, 2000.

Dror, M. Note on the complexity of the shortest-path models for column generation in VRPTW. *Operations Research*, vol. 42, no. 5, pp. 977–978, 1994.

Elhedhli, S.; Goffin, J.-L. The integration of an interior-point cutting plane method within a branch-and-price algorithm. *Mathematical programming*, vol. 100, pp. 267–294, 2004.

Elwes, R. The algorithm that runs the world. *New Scientist*, vol. 215, no. 2877, pp. 32–37, 2012.

Engau, A.; Anjos, M. F.; Vannelli, A. A primal-dual slack approach to warmstarting interior-point methods for linear programming. In: Chinneck, J. W.; Kristjansson, B.; Saltzman, M. J., eds. *Operations Research and Cyber-Infrastructure*, Springer, 2009, pp. 195–217.

Engau, A.; Anjos, M. F.; Vannelli, A. On interior-point warmstarts for linear and combinatorial optimization. *SIAM Journal on Optimization*, vol. 20, no. 4, pp. 1828–1861, 2010.

Falkenauer, E. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, vol. 2, pp. 5–30, 1996.

Feillet, D. A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR: A Quarterly Journal of Operations Research*, vol. 8, pp. 407–424, 2010.

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

Ford, L. R.; Fulkerson, D. R. A suggested computation for maximal multi-commodity network flows. *Management Science*, vol. 5, no. 1, pp. 97–101, 1958.

Fourer, R. Notes on the dual simplex method, Draft report, 1994.

Frangioni, A. Generalized bundle methods. *SIAM Journal on Optimization*, vol. 13, pp. 117–156, 2002.

Friedmann, O. A subexponential lower bound for Zadeh's pivoting rule for solving linear programs and games. *Integer Programming and Combinatoral Optimization*, pp. 192–206, 2011.

Friedmann, O.; Hansen, T.; Zwick, U. Subexponential lower bounds for randomized pivoting rules for the simplex algorithm. In: *Proceedings of the 43rd annual ACM symposium on Theory of computing*, ACM, 2011, pp. 283–292.

Galati, M.; Ralphs, T.; Wang, J. *Computational experience with generic decomposition using the dip framework.* Tech. Rep., Technical report, COR@L Laboratory, Lehigh University, 2012.

Gay, D. M. Eletronic mail distribution of linear programming test problems. *Lucent Bell Laboratories*, pp. 1–13, 1997.

Geoffrion, A. M. Lagrangean relaxation for integer programming. *Mathematical Programming Studies*, pp. 82–114, 1974.

Gilmore, P. C.; Gomory, R. E. A linear programming approach to the cutting-stock problem. *Operations Research*, vol. 9, no. 6, pp. 849–859, 1961.

Gilmore, P. C.; Gomory, R. E. A linear programming approach to the cutting stock problem - Part II. *Operations Research*, vol. 11, no. 6, pp. 863–888, 1963.

Goffin, J. L.; Haurie, A.; Vial, J. P. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, vol. 38, no. 2, pp. 284–302, 1992.

Goffin, J. L.; Vial, J. P. Convex nondifferentiable optimization: a survey focussed on the analytic center cutting plane method. *Optimization methods and software*, vol. 17, no. 5, pp. 805–867, 2002.

Gomory, R. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, 1958.

Gondzio, J. HOPDM (version 2.12) - a fast LP solver based on a primal-dual interior point method. *European Journal of Operational Research*, vol. 85, no. 1, pp. 221–225, 1995.

Gondzio, J. Multiple centrality corrections in a primal-dual method for linear programming. *Comput. Optim. Appl.*, vol. 6, no. 2, pp. 137–156, 1996.

Gondzio, J. Warm start of the primal-dual method applied in the cutting plane scheme. *Mathematical programming*, vol. 83, no. 1, pp. 125–143, 1998.

Gondzio, J. Interior point methods 25 years later. *European Journal of Operational Research*, vol. 218, no. 3, pp. 587–601, 2012.

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

Gondzio, J.; Grothey, A. Reoptimization with the primal-dual interior point method. *SIAM Journal on Optimization*, vol. 13, no. 3, pp. 842–864, 2003.

Gondzio, J.; Grothey, A. Direct solution of linear systems of size 10^9 arising in optimization with interior point methods. In: Wyrzykowski, R.; Dongarra, J.; Meyer, N.; Wasniewski, J., eds. *Lecture Notes in Computer Science*, 2006, pp. 513–252.

Gondzio, J.; Grothey, A. A new unblocking technique to warmstart interior point methods based on sensitivity analysis. *SIAM Journal on Optimization*, vol. 19, no. 3, pp. 1184–1210, 2008.

Gondzio, J.; Sarkissian, R. *Column generation with a primal-dual method.* Tech. Rep., Logilab, 1996.

Hall, J. A. J.; McKinnon, K. I. M. Hyper-sparsity in the revised simplex method and how to exploit it. *Computational optimization and applications*, vol. 32, no. 3, pp. 259–283, 2005.

Homberger, J.; Gehring, H. A two-phase hybrid metaheuristic for the vehicle routing problem with time windows. *European Journal of Operational Research*, vol. 162, no. 1, pp. 220–238, 2005.

Hu, J. F.; Pan, P. Q. An efficient approach to updating simplex multipliers in the simplex algorithm. *Mathematical programming*, vol. 114, pp. 235–248, 2008.

IBM ILOG CPLEX v.12.1 *Using the CPLEX callable library.* 2010.

Irnich, S.; Desaulniers, G. Shortest path problems with resource constraints. In: Desaulniers, G.; Desrosiers, J.; Solomon, M. M., eds. *Column Generation*, Springer US, pp. 33–65, 2005.

Irnich, S.; Villeneuve, D. The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, vol. 18, no. 3, pp. 391–406, 2006.

Jans, R.; Degraeve, Z. Improved lower bounds for the capacitated lot sizing problem with setup times. *Operations Research Letters*, vol. 32, no. 2, pp. 185 – 195, 2004.

Jepsen, M.; Petersen, B.; Spoorendonk, S.; Pisinger, D. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, vol. 56, no. 2, pp. 497–511, 2008.

John, E.; Yildirim, A. Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension. *Computational Optimization and Applications*, vol. 41, pp. 151–183, 2008.

Kallehauge, B.; Larsen, J.; Madsen, O. B. Lagrangian duality applied to the vehicle routing problem with time windows. *Computers and Operations Research*, vol. 33, no. 5, pp. 1464 – 1487, 2006.

Kallehauge, B.; Larsen, J.; Madsen, O. B.; Solomon, M. M. Vehicle routing problem with time windows. In: Desaulniers, G.; Desrosiers, J.; Solomon, M. M., eds. *Column Generation*, Springer US, pp. 67–98, 2005.

Karmarkar, N. A new polynomial-time algorithm for linear programming. *Combinatorica*, vol. 4, pp. 373–395, 1984.

Kelley, J. E., J. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 4, pp. 703–712, 1960.

Khachiyan, L. G. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, vol. 20, pp. 191–194, 1979.

Koberstein, A. *The dual simplex method, techniques for a fast and stable implementation.* Ph.D. thesis, University of Paderborn II, Berlin, Germany, 2005.

Koberstein, A. Progress in the dual simplex algorithm for solving large scale LP problems: techniques for a fast and stable implementation. *Computational Optimization and Applications*, vol. 41, pp. 185–204, 2008.

Koberstein, A.; Suhl, U. H. Progress in the dual simplex method for large scale LP problems: practical dual phase 1 algorithms. *Computational Optimization and Applications*, vol. 37, no. 1, pp. 49–65, 2007.

Koch, T. The final netlib-LP results. *Operations Research Letters*, vol. 32, pp. 138–142, 2004.

Kohl, N.; Desrosiers, J.; Madsen, O.; Solomon, M.; Soumis, F. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, vol. 33, no. 1, pp. 101–116, 1999.

Kostina, E. The long step rule in the bounded-variable dual simplex method: Numerical experiments. *Mathematical Methods of Operations Research*, vol. 55, pp. 413–429, 2002.

Land, A.; Doig, A. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pp. 497–520, 1960.

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

Lübbecke, M. Automatic decomposition and branch-and-price – a status report. *Experimental Algorithms*, pp. 1–8, 2012.

Lübbecke, M. E.; Desrosiers, J. Selected topics in column generation. *Operations Research*, vol. 53, no. 6, pp. 1007–1023, 2005.

Macedo, R.; Alves, C.; Valerio de Carvalho, J.; Clautiaux, F.; Hanafi, S. Solving the vehicle routing problem with time windows and multiple routes exactly using a pseudo-polynomial model. *European Journal of Operational Research*, vol. 214, no. 3, pp. 536–545, 2011.

Maros, I. A general phase-1 method in linear programming. *European Journal of Operational Research*, vol. 23, pp. 64–77, 1986.

Maros, I. *Computational techniques of the simplex method.* Kluwer Academic Publishers, 2003a.

Maros, I. A generalized dual phase-2 simplex algorithm. *European Journal of Operational Research*, vol. 149, pp. 1–16, 2003b.

Marsten, R. E.; Hogan, W. W.; Blankenship, J. W. The boxstep method for large-scale optimization. *Operations Research*, vol. 23, no. 3, pp. 389–405, 1975.

Martinson, R. K.; Tind, J. An interior point method in Dantzig-Wolfe decomposition. *Computers and Operation Research*, vol. 26, pp. 1195–1216, 1999.

du Merle, O.; Hansen, P.; Jaumard, B.; Mladenovic, N. An interior point algorithm for minimum sum-of-squares clustering. *SIAM J. Sci. Comput.*, vol. 21, no. 4, pp. 1485–1505, 1999.

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

Mitchell, J. E. Computational experience with an interior point cutting plane algorithm. *SIAM Journal of Optimization*, vol. 10, no. 4, pp. 1212–1227, 2000.

Mitchell, J. E.; Todd, M. J. Solving combinatorial optimization problems using Karmarkar's algorithm. *Mathematical Programming*, vol. 56, pp. 245–284, 1992.

Munari, P.; Gondzio, J. Using the primal-dual interior point algorithm within the branch-price-and-cut method. *Computers & Operations Research*, doi: 10.1016/j.cor.2013.02.028, 2013.

Munari, P. A. *Técnicas computacionais para a implementação eficiente e estável de métodos tipo simplex.* Master's thesis, Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, 2009.

Munari, P. A.; Arenales, M. N. Estudos computacionais sobre a influência da mudança de escala no método simplex. In: *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional*, SOBRAPO, 2009.

Nemirovski, A. S.; Todd, M. J. Interior-point methods for optimization. *Acta Numerica*, vol. 17, pp. 191–234, 2008.

Nocedal, J.; Wright, S. J. Nonlinear optimization. *Springer, New York*, 2006.

ORCHARD-HAYS, W. *Advanced linear programming computing techniques.* McGraw-Hill, 1968.

PAN, P.-Q. A primal deficient-basis simplex algorithm for linear programming. *Applied Mathematics and Computation*, vol. 196, pp. 898–912, 2008.

PAPARRIZOS, K.; SAMARAS, N.; STEPHANIDES, G. An efficient simplex type algorithm for sparse and dense linear programs. *European Journal of Operational Research*, vol. 148, pp. 323–334, 2003.

POGGI DE ARAGÃO, M.; UCHOA, E. Integer Program Reformulation for Robust Branch-and-Cut-and-Price Algorithms. In: *In Proceedings of the Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*, 2003, pp. 56–61.

PUREZA, V.; MORABITO, R.; REIMANN, M. Vehicle routing with multiple deliverymen: Modeling and heuristic approaches for the VRPTW. *European Journal of Operational Research*, vol. 218, no. 3, pp. 636 – 647, 2012.

RIGHINI, G.; SALANI, M. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, vol. 51, no. 3, pp. 155–170, 2008.

ROUSSEAU, L.-M.; GENDREAU, M.; FEILLET, D. Interior point stabilization for column generation. *Operations Research Letters*, vol. 35, no. 5, pp. 660–668, 2007.

SANTOS, F. A counterexample to the Hirsch conjecture. *Annals of Mathematics*, vol. 176, pp. 383–412, 2012.

SILVA, C. T. L.; ARENALES, M. N.; SOUSA, R. S. Métodos do tipo dual simplex para problemas de otimizaçao linear canalizados e esparsos. *Pesquisa Operacional*, vol. 27, no. 3, pp. 457–486, 2007.

SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, vol. 35, no. 2, pp. 254–265, 1987.

SOUSA, R. S.; SILVA, C. T. L.; ARENALES, M. N. Métodos do tipo dual simplex para problemas de otimizaçao linear canalizados. *Pesquisa Operacional*, vol. 25, no. 3, pp. 349–382, 2005.

SPOORENDONK, S. *Cut and column generation.* Ph.D. thesis, Technical University of Denmark, Department of Management Engineering, 2008.

SUHL, L. M.; SUHL, U. H. A fast LU update for linear programming. *Annals of Operations Research*, vol. 43, pp. 33–47, 1993.

SUHL, U. H.; SUHL, L. M. Computing sparse LU factorizations for large-scale linear programming bases. *ORSA Journal on Computing*, vol. 2, no. 4, pp. 325–335, 1990.

TRIGEIRO, W. W.; THOMAS, L. J.; MCCLAIN, J. O. Capacitated lot sizing with setup times. *Management Science*, vol. 35, no. 3, pp. 353–366, 1989.

VANDERBECK, F. Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem. *Oper. Res.*, vol. 48, pp. 915–926, 2000.

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

Villeneuve, D.; Desrosiers, J.; Lübbecke, M.; Soumis, F. On compact formulations for integer programs solved by column generation. *Annals of Operations Research*, vol. 139, pp. 375–388, 2005.

Wagner, H. M.; Whitin, T. M. Dynamic version of the economic lot size model. *Management Science*, vol. 5, no. 1, pp. 89–96, 1958.

Wentges, P. Weighted Dantzig–Wolfe Decomposition for Linear Mixed-integer Programming. *International Transactions in Operational Research*, vol. 4, no. 2, pp. 151–162, 1997.

Wright, S. J. *Primal-dual interior-point methods*. SIAM, Philadelphia, 1997.

Yildirim, E. A.; Wright, S. J. Warm-start strategies in interior-point methods for linear programming. *SIAM Journal on Optimization*, vol. 12, no. 3, pp. 782–810, 2002.