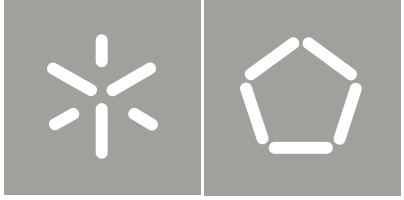

Universidade do Minho
Escola de Engenharia

Cláudio Manuel Martins Alves

Cutting and Packing: Problems, Models and Exact Algorithms

Tese submetida à Universidade do Minho para a obtenção do grau de Doutor no Ramo de Engenharia de Produção e Sistemas, área de Investigação Operacional

Trabalho efectuado sob a orientação de

Professor Doutor José Manuel Vasconcelos Valério de Carvalho
Departamento de Produção e Sistemas da Escola de Engenharia da Universidade do Minho

Fevereiro de 2005

Summary

Different integer programming models and exact algorithms for hard cutting and packing problems are addressed. We consider in particular the combinatorial problems of this family that are defined over a single dimension. The optimization procedures rely on tools from the field of integer programming, namely column generation, cutting planes and branch-and-bound. Integrating the three into the same algorithm is not straightforward, and has been used with some success only recently. The combined method is known as branch-and-price-and-cut. It requires a great part of customization, which is directly related to the specific problem that is being tackled.

We consider three variants of the standard one-dimensional Cutting Stock Problem, and its packing counterpart, the Bin-Packing Problem. We study the case for which more than a single large object is available, and an optimal cutting or packing has to be found so as to minimize the total length or capacity used. A related problem, which consists in selecting the best assortment of large objects subject to cardinality constraints, is also investigated. The problem of maximizing the homogeneity of the plans, and hence reducing the number of setups involved with its execution, is addressed. This combinatorial problem is commonly referred to as the Pattern Minimization Problem, and is particularly hard. We propose to improve an existing model, and describe how to derive new valid cutting planes for it using an original approach. Finally, we describe an exact solution algorithm for the Ordered Cutting Stock Problem. This problem has been formulated recently. It consists in finding the best assignment of small items to the stock rolls, avoiding breaks among pre-defined lots of items. Three integer programming models are presented, along with two families of cutting planes and their respective separation algorithms.

Different strategies are explored to improve the convergence of the column generation algorithm, when applied to one-dimensional cutting and packing problems. New dual cutting planes are presented, and we describe how existing ones can be extended to the whole branch-and-bound search tree, for a given branching scheme. Alternative methods based on model aggregation are also explored. Two strategies are proposed. The first is based on a simple row aggregation scheme, while the second relies on a more sophisticated double aggregation of rows and columns.

All the procedures proposed were coded and tested. Various computational ex-

periments were conducted to evaluate their performance, using, with this purpose, problem instances from the literature, and randomly generated instances. The results are presented and discussed throughout the thesis.

Résumé

Au long de cette thèse, nous étudions plusieurs problèmes de découpe et d'empaquetage dont la difficulté est reconnue. Nous considérons en particulier les problèmes combinatoires à une seule dimension. Les algorithmes d'optimisation que nous explorons s'appuient sur des méthodes du domaine de la programmation entière, notamment la génération de colonnes, la méthode des plans coupants et la méthode de séparation et évaluation. L'intégration de ces trois méthodes dans un même algorithme ne s'effectue pas directement. En fait, cette intégration n'a été réussie avec succès que très récemment. L'algorithme qui en résulte est connu sous le nom de méthode de séparation, génération de colonnes et plans coupants. Compte tenu du problème qui est abordé, beaucoup d'adaptations sont nécessaires pour parvenir à un algorithme performant.

Nous considérons ici les variantes des problèmes standards à une dimension de découpe et d'empaquetage. Nous étudions le cas dans lequel plusieurs objets de grande dimension sont disponibles, et une solution optimale doit être trouvée de manière à minimiser la largeur totale, ou capacité, utilisée. Nous explorons également le problème qui consiste à choisir le meilleur lot d'objets de grande dimension quand il existe des restrictions de cardinalité. La maximisation de l'homogénéité des solutions de découpe, et correspondante minimisation du nombre de changements de patrons de découpe, sont également étudiées. Ce problème est connu sous le nom de Problème de Minimisation des Patrons de Découpes, et il est particulièrement difficil. Nous montrons comment un modèle qui a été récemment proposé peut être amélioré, et nous expliquons comment dériver des inégalités valides en utilisant une approche originale. Finalement, nous décrivons un algorithme de résolution exacte pour le problème de découpe ordonnée. Ce problème a été formulé récemment. Il consiste à trouver la meilleure association des petits et grands objets en évitant qu'il y ait des cassures entre les lots qui ont été prédéfinis. Nous proposons trois modèles de programmation entière, ainsi que deux familles de plans coupants et leurs algorithmes de séparation.

Plusieurs stratégies ayant pour but d'améliorer la convergence de la méthode de génération de colonnes sont explorées. Nous considérons le cas particulier où cette méthode est appliquée à des problèmes de découpe et d'empaquetage à une dimension. Nous présentons de nouveaux plans coupants dans l'espace dual, et nous décrivons comment étendre d'autres abordages présentés dans la littérature scientifique à l'arbre

généré par la méthode de séparation et d'évaluation, en assumant un schéma de séparation spécifique. Des méthodes alternatives basées sur l'aggrégation de modèles sont aussi explorées. Deux méthodes sont proposées. La première est basée sur un schéma simple d'aggrégation de restrictions, tandis que la seconde s'appuie sur une aggrégation plus sophistiquée de variables et de restrictions.

Toutes les procédures proposées ont été programmées et testées. Plusieus expériences computationelles ont été menées pour évaluer leur performance en utilisant, à cet effet, des instances décrites dans la littérature, et des instances générées aléatoirement. Nous présentons les résultats obtenus, et nous les discutons tout au long de la thèse.

Resumo

Nesta tese, são apresentados diferentes modelos de programação inteira e algoritmos de resolução exacta para problemas difíceis de corte e empacotamento. Consideramos em particular os problemas combinatórios dessa família numa única dimensão. Os algoritmos de optimização que exploramos assentam em métodos do domínio da programação inteira, nomeadamente a geração de colunas, planos de corte e o método de partição e avaliação sucessivas. Integrar esses três métodos no mesmo algoritmo não é um exercício directo. Na realidade, isso foi conseguido só muito recentemente. O algoritmo resultante é designado de partição, geração de colunas e corte. Atendendo ao problema que é abordado, várias adaptações têm de ser efectuadas.

Consideramos três variantes dos problemas standards de corte e empacotamento com uma dimensão. Estudamos o caso para o qual mais de um tipo de rolos ou contentores estão disponíveis, e uma solução óptima tem de ser encontrada de forma a minimizar a largura ou capacidade total utilizada. Investigamos também um problema relacionado no qual deve ser escolhido o melhor lote de rolos ou contentores atendendo a restrições de cardinalidade. O problema no qual se pretende maximizar a homogeneidade das soluções de corte ou empacotamento, e dessa forma minimizar o número de mudanças que ocorrem quando se executa o plano, é analizado. Esse problema combinatório é também conhecido por Problema da Minimização de Padrões, e é de difícil resolução. Propomos melhorar um modelo existente, e descrevemos como derivar planos de corte válidos usando uma abordagem original. Finalmente, descrevemos um algoritmo de resolução exacta para o problema de corte ordenado. Esse problema foi formulado muito recentemente. Consiste em determinar a melhor afectação de itens aos rolos, evitando interrupções entre lotes pre-definidos de itens. Três modelos de programação inteira são apresentados, juntamente com duas famílias de planos de corte e os seus respectivos algoritmos de separação.

Diferentes estratégias para melhorar a convergência do método de geração de colunas são exploradas. Consideramos o caso no qual esse método é aplicado a problemas de corte e empacotamento a uma dimensão. Novos cortes duais são apresentados, e descrevemos como cortes que já foram descritos na literatura podem ser extendidos à totalidade da àrvore de partição, assumindo um esquema de partição específico. Métodos alternativos baseados em agregação de modelos são também explorados. Duas

estratégias são propostas. A primeira é baseada num esquema simples de agregação de restrições, enquanto a segunda assenta num esquema mais sofisticado de restrições e variáveis.

Todos os procedimentos propostos foram codificados e testados. Várias experiências computacionais foram levadas a cabo, usando, para isso, instâncias descritas na literatura, e instâncias geradas aleatoriamente. Os resultados são apresentados, e discutidos ao longo da tese.

Acknowledgments

The concretization of a project is rarely the effect of a single force. My case is not an exception. I am grateful to many persons and institutions, who played an important role, direct or indirectly.

First of all, my PhD supervisor, Professor Valério de Carvalho. I want to thank him for the numerous opportunities he provided me. I can not, and do not, forget the precious help he gave me right at the beginning of this project, a few years ago. I would like to thank him for sharing with me his enthusiasm for the field, for allowing me to present my work on numerous national and international conferences. As all my PhD colleagues, surely, I appreciated his constant availability. During my years of post-graduate formation, he has been a real source of stimulus. In the future, he will be for me the example of how a PhD supervisor should be.

I want to thank also all my colleagues of the Department of Production and Systems Engineering, specially those of the Operations Research Group, for their suggestions, for their friendship.

I am grateful to the University of Minho, and its School of Engineering, for providing me the conditions to realize this work, among which are the three years during which I was exempted from teaching obligations.

I want to thank the Educational Development Program for Portugal, FSE/PRODEP Doutoramentos (Concurso n^{o}2/5.3/PRODEP/2001), for their grant.

Finally, my private dream team. I want to thank Sónia, surely one of my greatest supports, for her understanding and encouragement. I want to publicly apologize to my sister for enduring my jokes, and thank her for her company. Finally, I wish to thank my parents. It will take too long to enumerate all the reasons that motivate my gratitude for them, if it is even possible to enumerate them all. I hope only to be able someday to reward their efforts and sacrifices.

Braga, February 2005

Cláudio Alves

In memory of my grandparents,

To my mother, Beatriz,

To my father, João.

Chercheurs que le néant captive,
Qui, dans l'ombre, avons en passant
La curiosité chétive
Du ciron pour le ver luisant,

Poussière admirant la poussière,
Nous poursuivons obstinément,
Grains de cendre, un grain de lumière
En fuite dans le firmament!

Victor Hugo, *Les Contemplations*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Scope of the Thesis

Many are the experiences that attest the power of mathematical programming tools, in the most diverse areas. In fact, there is not a clear frontier delimiting the application domain of mathematical programming. There are reports of applications in different types of industry, health care, education, computer sciences, services, logistics, among others. In all these contexts, we always find the same objective: to take the best possible decisions under various operational constraints. Applying mathematics with this purpose, modeling the problems so that they can be tackled by solution procedures adapted to the complexity of the cases, developing these algorithms to ensure an efficient and effective resolution of the problems, all of this has been approached with notable success by optimization practitioners. The considerable increase in the computing capacities played an important part in this success. It is now easier to manage the large volumes of data that are usually associated with real world problems, and this is clearly not the only benefit.

The solution approaches investigated in this thesis belong to the field of mathematical programming. We consider in particular problems in which the decisions involved are discrete, modeled by means of integer variables, and solved using integer programming methods. We restrict our study to exact algorithms whose aim is to find a proven optimal solution. However, this does not exclude the reference, in some parts of the text, to heuristic procedures, which are frequently a very good instrument that helps in improving the efficiency of the whole algorithms. A specific application domain is addressed here, namely, the cutting and packing area. Apart from their practical relevance, cutting and packing problems have been used since long to test different algorithmic approaches. The corresponding integer formulations that have been proposed for the general problems can be used to model many practical activities and processes.

Column generation, cutting planes and branch-and-bound are general algorithmic

tools commonly used to solve large scale integer programming problems. The development of a solution algorithm that aggregates these techniques depends on the particular case that is being studied. It must take into account different aspects of the underlying models, such as their structure. Along this thesis, different algorithms are proposed to optimally solve cutting and packing problems with a single dimension, and variants of the general problem. We describe alternative integer programming models, and explore ways of improving the efficiency of the solution procedures. All the algorithms proposed have been implemented, and tested on problem instances from the literature, and randomly generated instances. We report and discuss the results obtained.

1.2 Motivation and Objectives

Optimization algorithms combining column generation, cutting planes and branch-and-bound are referred to as branch-and-price-and-cut algorithms. Developing such algorithms is not straightforward. In fact, they began to be used together not a long time ago, and they have deserved in the last years a real investment of the research community. More and more problems are being tackled with these techniques, but they are still many open problems. The cutting and packing area is an example. There are practical variants of the standard problems, which are of great interest, and for which there is no reported attempts of solving them using branch-and-price-and-cut, or that have not been yet satisfactorily solved using these techniques. Hence, our aim was to investigate some of these problems, among which are the Multiple Length Cutting Stock Problem, the Pattern Minimization Problem, or also the Ordered Cutting Stock Problem, to analyze their structure, to explore alternative models, to derive families of valid cutting planes, and to finally study the behavior of the resulting algorithms by converting them into pieces of software.

There are also inherent weaknesses related to the general techniques addressed in this thesis, namely concerning column generation. It is well known that the column generation algorithm tends to converge slowly, with the optimal solution improving very slightly in the last iterations. Many results have been reported in the literature on attempts to overcome this undesirable characteristic. Some of the solutions proposed were able to accelerate column generation considerably. However, they were studied only within restricted contexts, for the linear relaxations of the models. When one is searching for an optimal integer solution through branch-and-bound, it would be much more interesting if he could used them along the whole branch-and-bound tree. In this thesis, this subject is explored. We also analyze alternative techniques, based on an original approach, to improve the convergence of the column generation algorithm, when applied specifically to cutting stock and bin-packing problems. Indeed, these problems have peculiarities that have never been explored, and which can be very

useful to accelerate column generation.

1.3 Outline

The present thesis is divided in 7 independent chapters, including the present one. Each chapter is self contained. The work described in each one of them is presented using a research paper writing style, some parts having already been submitted to scientific journals. As a consequence, the same subject may be treated in different points of the text. However, this happens only occasionally.

In Chapter 2, we briefly recall some aspects related to the solution techniques adopted, and to the family of problems tackled in this thesis, namely cutting and packing problems. The origin of the column generation algorithm is discussed, the Dantzig-Wolfe decomposition is described, the relation with lagrangean relaxation is presented, and the difficulties associated to the implementation of a branch-and-price-and-cut algorithm are analyzed. Cutting and packing is a vast application area. We refer to some of the problems that have been studied in the literature, and discuss the classification systems that were suggested so as to organize the numerous contributions that have been given so far. We proceed by reviewing the standard problem, which is the base of the work presented in this thesis. We present the integer programming models that were proposed in the literature, and we refer to some of the solution algorithms that have been developed.

Chapter 3 is devoted to the Multiple Length Cutting Stock Problem, and its packing counterpart, the Variable Sized Bin-Packing Problem. We present a branch-and-price-and-cut algorithm, which is based on two integer programming models from the literature that have never been combined in the same procedure. Many computational results are reported, comparing our approach to the few alternative exact algorithms described in the literature. We will see that our method clearly outperforms some of these algorithms. Additionally, the assortment problem is tackled. In practice, the availability of more than a single stock length frequently gives rise to the problem of selecting a restricted subset of lengths to order. We extend our algorithm to the case where the best assortment must be found so as to simultaneously ensure the minimum trim loss, and we analyze its behavior using various randomly generated instances.

In Chapter 4, we address the question of the convergence of column generation algorithms, within the scope of one-dimensional cutting and packing problems. We present new dual-optimal inequalities for the problem with multiple lengths, and prove their validity. Assuming a branching scheme similar to the one introduced in Chapter 3, we also give the conditions under which a family of valid dual inequalities for the Cutting Stock Problem, and Bin-Packing Problem, remains valid for a problem in which certain branching constraints have been enforced. These results allow to use a class of

dual cuts, which are proven to be very effective in accelerating the resolution of the pure linear relaxation by column generation, in the whole branch-and-bound search tree. An alternative approach to column generation stabilization is also explored. We try to take advantage of the prior knowledge concerning the properties of the optimal dual solutions of the Cutting Stock Problem. Our proposals are based on model aggregation. A single scheme is first tested, in which only the rows of the linear models are aggregated. The problem is solved with a two-phase column generation algorithm. A more sophisticated procedure is then investigated, relying on a double aggregation of the models. We present this aggregation scheme as an alternative way of enforcing dual constraints. Computational results are given, for many instances from the literature, and randomly generated ones.

Chapter 5 is devoted to the Pattern Minimization Problem. We explore a column generation model that has been recently proposed for this problem, and analyze the impact of restricting the set of admissible columns both in strengthening the model, and in improving the solution algorithm. We also use dual feasible functions to generate cutting planes in order to strengthen a model, which is known to be not very strong. As far as we know, this is the first reported attempt in deriving cutting planes for the knapsack polytope using these functions. To further strengthen the models, at each node of the branch-and-bound tree, we experiment deriving the cutting planes from surrogate constraints, obtained by combining different types of inequalities. Finally, the Pattern Minimization Problem is solved with a branch-and-price-and-cut algorithm, and computational results are reported to measure its efficiency. The algorithm was also extended to cope with multiple stock lengths.

In Chapter 6, we explore the Ordered Cutting Stock Problem, a problem for which no exact solution algorithms have ever been proposed. The problem is characterized by the existence of lots of items that must be cut without any interruption, being a single stack of end products open at any time. Three integer programming models are proposed: an assignment model, a flow model and a column generation reformulation. The latter is strengthened using two families of cutting planes: subtour elimination constraints and comb inequalities. The details of the pricing subproblem are given, as long as other aspects related to the final branch-and-price-and-cut algorithm. Random instances were generated, and used to test the algorithm. Our computational results are reported at the end of the chapter.

In Chapter 7, we will finish with some final concluding remarks, and give some directions for future research.

Chapter 2

Column Generation and 1-Dimensional Cutting and Packing

In the field of integer and combinatorial optimization, column generation is for the time being one of the most efficient solution technique. This optimization method emerged at the end of the fifties, and since then it experienced different degrees of interest. To identify the different periods related to column generation, nothing might be better that referring to the pioneering papers that gave rise to them. In 1958, the paper from Ford and Fulkerson [46] inspired the theory that would be formalized later by Dantzig and Wolfe, and referred to as the Dantzig-Wolfe decomposition method. Fundamentally, the proposal was to solve linear programs by using explicitly only a fraction of the formulation's data. Independently, Gilmore and Gomory [52] presented an application in the Cutting Stock Problem; their paper would become one of the most cited paper in the field of cutting stock problems. Many other application papers followed in the same decade. In 1984, Desrosiers *et al.* [32] proposed an algorithm for the vehicle routing problem with time windows, combining successfully column generation with branch-and-bound, a technique that became known as branch-and-price. Research on exact algorithms based on branch-and-price began to intensify, enlarging the spectrum of applications. Nowadays, many efforts are directed towards the resolution of some of the shortcomings inherent to column generation methods, like accelerating convergence.

The objective of this chapter is essentially to describe the context in which the work presented in the subsequent chapters evolved, and identify the general difficulties we must overcome when we want to develop and implement an algorithm based on column generation, branch-and-bound and cutting planes methods.

Keywords: Column Generation, Branch-and-Bound, Cutting Stock Problems

2.1 Large Scale Optimization Problems

Optimization problems that can be expressed as linear or integer linear programs are said to be large because their translation into this mathematical representation is done at the cost of a huge number of variables and/or constraints. To deal with such problems, one has to take into account all the singularities that may characterize them. For example, in practice, the density of the coefficients matrices for these problems is typically low. What we observe is that activities or products compete for a restricted set of resources, and, hence, many of the coefficients in the constraints are zeroes. Obviously, this data has not to be stored.

A model can be large because the original problem already involves a considerable number of products, activities or resources, or it can be large because one is using an alternative reformulation that depends on the complete enumeration of alternative solutions. These latter models can usually be obtained through a decomposition procedure, and for practical reasons they are frequently approached using column generation frameworks. Decomposition begins with a compact formulation, and divides the set of constraints into a set of "easy" constraints, and a set of "hard" constraints. Reformulation results in a model with less rows, but far more columns. This distinction among the parts of a model is a matter that is directly related to the notion of structure.

Large scale linear programs raise essentially two problems. The first consists in determining how to deal with a huge volume of data that, even if it can be compacted and even if the price of computing resources tends to decrease in a regular basis, still constitutes a real constraint in view of the actual computing capacities. The second concerns the available solution algorithms. For linear programs, the preferred solution method is usually the simplex algorithm. Variables that may possibly improve the current solution are found by pricing out non-basic columns, and choosing one with a negative reduced cost. This pricing scheme is based on enumeration, and, for models with an exponential number of columns, it is not computationally viable.

2.1.1 Structure

Certain models can be divided into independent blocks, which are in turn coupled by constraints, or alternatively by variables (or both). These blocks may be composed by a set of constraints that may allow them to be solved by dedicated algorithms more efficiently. For example, we may have blocks comprising the constraints of a shortest path problem, or a knapsack problem, for example. We can schematize as follows two of the most common structures found in LP models, those that are within the scope of our present research.

$$
\begin{array}{lllllllllll}
\text{(BD1)} & \min z = & c_1x_1 & + & c_2x_2 & +\ldots+ & c_nx_n & & & \\
& \text{subject to} & A_1^0x_1 & + & A_2^0x_2 & +\ldots+ & A_n^0x_n & \geq & b_0 \\
& & A_1^1x_1 & & & & & \geq & b_1 \\
& & & & A_2^2x_2 & & & \geq & b_2 \\
& & & & & \ldots & & & \\
& & & & & & A_n^nx_n & \geq & b_n \\
& & x_1 \geq 0, & & x_2 \geq 0, & \ldots & x_n \geq 0. & &
\end{array}
$$

$$
\begin{array}{lllllllllll}
\text{(BD2)} & \min z = & c_1x_1 & + & c_2x_2 & +\ldots+ & c_nx_n & + & dy & & \\
& \text{subject to} & A_1x_1 & & & & & + & B_1y & \geq & b_1 \\
& & & & A_2x_2 & & & + & B_2y & \geq & b_2 \\
& & & & & \ldots & & & & & \\
& & & & & & A_nx_n & + & B_ny & \geq & b_n \\
& & x_1 \geq 0, & & x_2 \geq 0, & \ldots & x_n \geq 0, & & y \geq 0. & &
\end{array}
$$

The independence of the blocks leads to matrices that exhibit a diagonal structure, with additional coupling elements. In BD1, the coupling is made by constraints that interrelate the whole set of variables, while in BD2 this coupling is done with variables with positive coefficients in most of the constraints. Note that the dual of BD2 is a problem that has the same structure as BD1, and hence, the approaches developed for BD1 can be extended to the dual of BD2 as well. This structure is commonly referred to as dual angular [80].

The structure of the coefficient matrices suggests a hierarchical solution approach, in which the independent blocks and the coupling elements are treated separately. This separation is interesting only if the problems related to the blocks are easier to solve alone than together with the other constraints. At a higher level, the coupling constraints are considered in a problem that is often called the master problem. In column generation algorithms, information flows between the two levels. The blocks are solved independently, contributing with a solution that is gathered and evaluated in the master.

Structure is formally exploited by methods based on decomposition techniques, such as the Dantzig-Wolfe decomposition, and methods based on Lagrangean relaxation. In certain cases, the reformulations provide better lower bounds on the integer optima than the ones obtained with the linear relaxations of the original compact models.

2.1.2 Example: the Cutting Stock Problem

The cutting stock problem is an example of such problems with a diagonal structure, with blocks of constraints that characterize other related and well-known combinatorial

problems. Kantorovich [76] was the first to propose an integer linear programming formulation, based on assignment variables as follows.

$$\min \quad \sum_{j=1}^{n} y_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} x_{ij} \geq b_i, \ i = 1, \ldots, m, \tag{2.2}$$

$$\sum_{i=1}^{m} w_i x_{ij} \leq W_j y_j, \ j = 1, \ldots, n, \tag{2.3}$$

$$y_j \in \{0, 1\}, \ j = 1, \ldots, n, \tag{2.4}$$

$$x_{ij} \geq 0 \text{ and integer}, \ i = 1, \ldots, m, \ j = 1, \ldots, n. \tag{2.5}$$

Model (2.1)-(2.5) has binary and general integer variables, with y_j representing the choice of roll j, and x_{ij} the number of items of size w_i assigned to roll j. Here, the blocks consist in the knapsack constraints (2.3), one for each roll, while the demand constraints (2.2) work as the linking constraints. It is well known that knapsack problems are NP-hard [49]. However, they can be solved very efficiently in practice [88], and hence, cutting stock problems have been traditionally approached using decomposition based procedures that take advantage of this special structure.

Model (2.1)-(2.5) also grows very quickly in size. For a small instance with 10 different item sizes and 100 available rolls, the model has 1100 variables and 110 constraints, while for an instance with 100 item sizes and 1000 rolls, the number of variables goes to 101000 for 1100 constraints. Obviously, it is possible to reduce the size of the model by computing a better upper bound on the optimum solution cost. This model has also some practical drawbacks, such as symmetry and a rather poor lower bound, that make the decomposition based reformulations much more attractive. Here, symmetry is characterized by the fact that exchanging items between two rolls leads to solutions that are different in terms of variables' values, but absolutely the same in practice.

2.2 Dantzig-Wolfe Decomposition

2.2.1 Principle

The Dantzig-Wolfe decomposition procedure described in [25] applies to linear programs with the following form

$$\begin{aligned} (LP) \qquad \min \quad & cx \\ \text{subject to} \quad & Ax \geq b, \\ & x \in X, \\ & x \in \mathbb{R}^n_+. \end{aligned}$$

The principle is to replace the second set of constraints by an alternative representation based on an enumeration of the extreme points and extreme rays of X, using for this purpose the theorem of Minkowski. Indeed, this theorem states that the points of the nonempty polyhedron X can be expressed as a convex combination of its extreme points (set P) and a non-negative combination of its extreme rays (set R), *i.e.*,

$$X = \left\{ x \in \mathbb{R}^n_+ : \sum_{i \in P} \lambda_i P_i + \sum_{i \in R} \lambda_i R_i,\ \sum_{i \in P} \lambda_i = 1,\ \lambda_i \geq 0,\ \forall i \right\},$$

or simply as a convex combination of its extreme points if the polyhedron is bounded. After the substitution, the λ_i coefficients become the new decision variables. However, since the number of extreme points and rays is generally exponential, the reformulation will also have an exponential number of columns. For this reason, the original formulation is also referred to as the compact formulation [126].

The coefficient matrix for the constraints that characterizes the polyhedron X in the linear program may not have any special property, like unimodularity [105], and hence, the extreme points of X may be fractional. In the combination formulae, if one uses a set of integer points of X instead, the LP bound of the resulting reformulation might be better than the one provided by the compact formulation [31]. This is what happens with the cutting stock problem. Solving the LP relaxation of the knapsack problems does not ensure that one will get an integer solution. Alternatively, solving the integer knapsack subproblems to optimality yields a reformulation with a continuous bound that is known to be very tight [85].

In [9], the authors give additional reasons that may convince one to prefer these reformulations. Among them, the problem of symmetry is pointed out. For example, for the cutting stock problem, the Dantzig-Wolfe decomposition leads to a model where the reference to the specific roll to which the items are assigned is removed. This eliminates the inherent symmetry of the compact model (2.1)-(2.5).

To develop their decomposition algorithm for linear programs [25], Dantzig and Wolfe were inspired by the previous work on multicommodity network flows by Ford and Fulkerson. The oldest reported papers applying this decomposition are [52, 53, 5].

The decomposition of a linear program leads to a reformulation that is itself composed by non integer variables. When the original problem is in fact an integer problem, integrality must be enforced by some ways, other than forcing the multipliers of the combination to be integer, since the original Dantzig-Wolfe decomposition considers the convex hull of X, and not the convex hull of the integer points in X. Vanderbeck [124] proposed an alternative based on discretization for the decomposition of integer programs, based on the principle that the elements of $X \cap \mathbb{Z}^n$ can be expressed as a combination of integer extreme points (P) and integer extreme rays (R) as follows

$$X \cap \mathbb{Z}^n = \left\{ x \in \mathbb{R}^n_+ : \sum_{i \in P} \lambda_i P_i + \sum_{i \in R} \lambda_i R_i,\ \sum_{i \in P} \lambda_i = 1,\ \lambda_i \geq 0,\ \text{and integer},\ \forall i \right\}.$$

The result is a reformulation that is now a true integer program.

2.2.2 Lagrangean Relaxation

There is a close relation between the formulations obtained with the decomposition method described above and those obtained by lagrangean relaxation. For an integer program with the same form as the linear formulation LP given in the previous section, the lagrangean problem obtained by dualizing the first set of constraints is defined as follows

$$\begin{aligned} z_{LR}(\lambda) = \min \quad & cx + \lambda^T(b - Ax) \\ \text{subject to} \quad & x \in X, \\ & x \in \mathbb{Z}^n_+. \end{aligned}$$

Here, the set of Lagrange multipliers λ are nonnegative. Maximizing the optimum $z_{LR}(\lambda)$ of the lagrangean problem over all the feasible values of λ gives rise to a so called lagrangean dual problem.

If we assume that the polyhedron X is bounded, and substitute in the lagrangean dual problem the constraints that define X by a convexification based on its extreme points, we get a model whose dual is precisely the one obtained with a Dantzig-Wolfe decomposition in which only $Ax \geq b$ is kept in the master. Besides this correspondence between the models, there is obviously a relation between the lower bounds, as it was first pointed out by Geoffrion in 1974 [51]. Regarding the quality of the lower bound provided by the lagrangean dual problem compared to the continuous bound of the original formulation, similar conclusions apply, in the sense that only a lagrangean problem that has not the integrality property can lead to an improved lower bound.

2.2.3 Column Generation Algorithm

While the Dantzig-Wolfe decomposition procedure is a reformulation technique, column generation is the practical optimization method devised to solve the resulting models. As we already noted, the huge number of columns in these reformulations makes the use of solution methods such as simplex on the complete models impractical, even for moderate size problems. With column generation, one deals with only a partial set of columns at each iteration. In fact, this method allows the enumeration of columns to be done implicitly, as the revised simplex method already does. The pivoting step of the simplex method only updates the B^{-1} matrix, and access to the nonbasic columns is required only when one has to price out attractive columns. At each step of the column generation procedure, only a portion of the whole set of columns is kept in the master. Pricing is done dynamically, taking advantage of the prior knowledge of the structure of the columns.

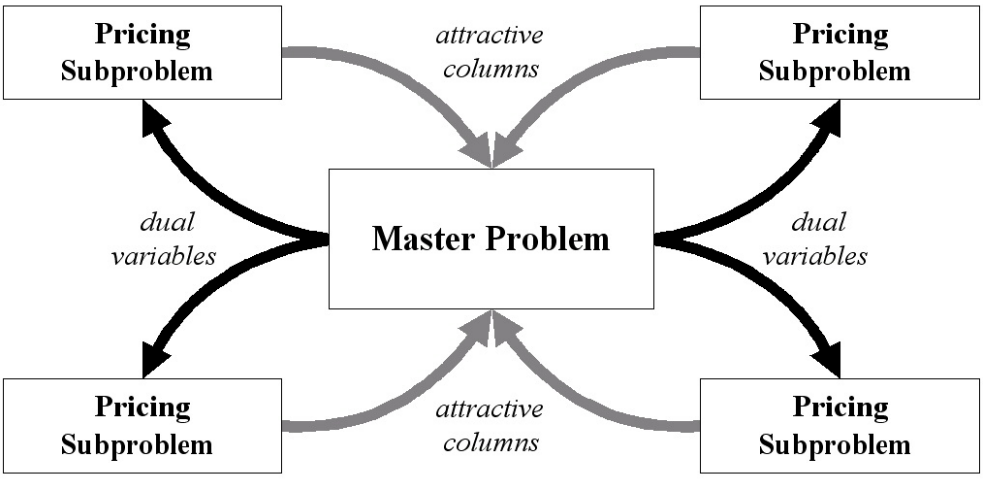

Figure 2.1: Outline of column generation algorithms

The elements of a column generation algorithm can be schematically divided in two parts: a restricted master problem, and a set of independent pricing subproblems. These two components interact with one another. Based on a restricted set of columns, the master is used to update the dual variables, so as to correctly form the cost formulae to price the nonbasic columns. In turn, the pricing subproblems use this data to optimize one or more auxiliary problems in order to find the best nonbasic columns, or a set of such columns. These subproblems are independent, and are related to the blocks discussed above. In Figure 2.1, we depict the interaction that occurs in a column generation algorithm.

Both the column generation algorithm and the subgradient method, which is commonly used to solve lagrangean dual problems, are methods based on the iterative update of the vector of dual variables, or dual multipliers. However, column generation is now being usually preferred to lagrangean relaxation, since it takes advantage of the whole information in a better way, yielding algorithms that are robust, and overcoming the considerable instability of the dual variables and the convergence problems usually associated to lagrangean relaxation. Recent developments in LP software helps in making the update of the dual variables, when reoptimizing the LP, a not so heavy burden.

In his technical review [128], Wilhelm divides the family of column generation algorithms presented so far in three groups. In the first group, the author puts all the implementations in which a large set of columns is generated with an auxiliary model, added to the master, which is in turn optimized. The essential feature of these algorithms is that the master receives columns only once. As an example, we can cite the paper of Hoffman and Padberg [66]. In the second group, Wilhelm gathers all the column generation algorithms in which there is an iterative interaction between the master and the subproblems, but which are not based on an explicit Dantzig-Wolfe

decomposition of a compact model. The papers from Gilmore and Gomory [52, 53] are in this category. At last, we have those column generation algorithms which are effectively based on a Dantzig-Wolfe decomposition, as for example [5, 6].

There is a nice economic interpretation associated to the Dantzig-Wolfe decomposition, and the related solution method, column generation. The whole framework can be seen as a decentralized decision system. At the top, the master, or coordinator, has to plan the global operation, managing a set of limited resources. For this purpose, he/she proposes to the independent subsystems a set of shadow prices to measure the attractiveness of their products, or activity vectors. The subsystems then reply with an offer. The master gathers the new proposals, evaluates them, and derives a new set of prices. And the process repeats until equilibrium is reached, *i.e.*, the subsystems are no more able to give any interesting proposal, given the prices they receive [80].

2.3 Restricted Master Problem

The master problem is said to be restricted because it does not comprise all the feasible columns. This definition raises immediately a first problem: how to initialize this master problem? A further question concerns the way the columns should be managed, or even how to solve the master problem. In [83], Luebbecke and Desrosiers give a brief review of some possible solution methods that have been proposed. In the sequel, we recall some aspects related to the solution of master problems.

2.3.1 Initialization and Columns Management

Initializing the master problem with an auxiliary column that has a high cost, and no meaning in the context of the practical problem to solve, ensures that there will always be a feasible basis for the simplex method. This artificial column may be necessary right at the beginning of the solution process, because there is no more columns in the master, or at a node of a branch-and-bound search tree, because the respective problem with the newly added branching constraint is infeasible. The master problem can be further initialized with a pool of columns obtained with a heuristic. These columns surely allow column generation to begin with a solution of smaller cost, but they do not guarantee that column generation will converge faster.

At each step of the column generation process, naturally, the size of the master becomes larger. In [9], Barnhart *et al.* suggest removing the nonbasic columns that have a very negative (very positive, when the master is a minimization problem) reduced cost.

Regarding the columns that should be added to the master problem, some experiences are reported in the literature. As pointed out in [9], results from [121] indicate

that when pricing subproblems are hard to solve, columns with negative reduced costs may be searched for with heuristic procedures, and one should resort to optimization only if they fail in identifying at least one of these columns. For easy pricing subproblems, exact solutions must be used alternatively. Gilmore and Gomory [53] compared the computing times solving the subproblems at optimality and heuristically, and concluded that the first strategy leads consistently to less pivoting operations in the master, and faster convergence.

2.3.2 Cutting Planes

The continuous bound provided by the restricted master problem is essential in algorithms combining column generation and branch-and-bound based on linear programs. The quality of the bounds of reformulations obtained through Dantzig-Wolfe decomposition is usually good, but sometimes it is not enough to ensure the success of branch-and-price algorithms. The Pattern Minimization Problem [123] is an example. By adding valid cutting planes to the master, one can get a stronger model. However, caution must be taken since, to report the dual values to the pricing subproblems, the original structure of these subproblems may be destroyed.

A fundamental aspect of column generation algorithms is the relative easiness with which the pricing subproblems can be solved. Hard subproblems must be avoided. Even if they can lead to stronger restricted master problems, allowing one to prune earlier a branching node, or to find an integer solution faster, the fact that they have to be solved many times would make the whole algorithm clearly inefficient. Hence, cutting planes must be chosen in order to keep the subproblems tractable. A possible way of preserving the tractability of the subproblems is to derive cuts based on the variables of the original formulation. In [118], Van den Akker *et al.* give a formal proof showing the accuracy of this statement, when the structure of the cost coefficients does not influence the algorithm for the solution of the pricing subproblem.

Deriving valid cutting planes for the master is also referred to as row generation. This process is dual to the column generation one. Examples of papers reporting on attempts to combine row and column generation are [6, 92, 8].

2.3.3 Convergence

Solving master problems with column generation generally involves many iterations. In fact, these processes exhibit a long tail convergence, characterized by a value of the optimum which is only marginally improved in the last iterations. In [78], Kim and Nazareth classify the difficulties suffered by column generation algorithms in two groups: combinatorial and numerical. In the former, the combinatorial structure of the subproblems' facets explains the difficulty in finding the optimal convex combina-

tion of extreme points. In the latter, it is the nature of the columns enumerated in the restricted master problem that causes numerical instabilities. In their paper, the authors give as an example the case of badly scaled columns.

Degeneracy in linear programs is another combinatorial difficulty that is frequently pointed out as a major cause of the slow convergence of the column generation processes. When degeneracy occurs, many pivoting operations are necessary for the value of the master problem to begin improving again, and hence, many columns have to be generated for nothing more than leaving a degenerate extreme point. Furthermore, we observe in practice large fluctuations of the dual variables. This behavior leads to the generation of different columns, many of them without any contribution to the optimal solution.

Different approaches have been devised to improve the convergence of column generation algorithms. We will review these methods later in Chapter 4, which is fully dedicated to the issue of convergence.

2.4 Pricing Subproblem

In the simplex method, pricing has a double objective: to verify that the current basic solution satisfies the optimality conditions, and to select a potentially improving variable, if this is not the case. In column generation algorithms, pricing out nonbasic columns that do not belong to the restricted master is done through the resolution of another optimization problem. The way these subproblems are solved depends on their particular structure. In [70], the min-cut clustering problem is solved using a set partitioning master problem, and a mixed integer programming subproblem denoted by the authors as the knapsack quadratic program. This pricing subproblem is solved by combining cutting planes with branch-and-bound. However, for pricing subproblems with special structure, like the shortest path problem which arises frequently from Dantzig-Wolfe decompositions [118, 8], there are usually better algorithms. For subproblems that are NP-hard, this situation might still apply. For example, the knapsack problem for cutting stock problems can be solved efficiently with pseudo-polynomial algorithms [88, 94]. Constrained versions of the shortest path problem are also solved as pricing routines for routing problems. For the problem with time windows, efficient algorithms are given in [31].

Kim and Nazareth [78] suggest solving the pricing subproblems with interior point methods. The authors claim that using points within the interior of the subproblem polytope together with some of its extreme points and extreme rays can help in reducing the difficulties that lead to the slow convergence of column generation algorithms.

There are different aspects related to the implementation of a pricing scheme. The first concerns the space of variables to inspect (fully or partially), and the number of

attractive columns one wants to generate, and to add to the restricted master (single or multiple). A possible way of implementing a partial pricing scheme is to solve the independent subproblems sequentially until an attractive column is found. On the other hand, multiple pricing, that consists in adding more than one column per iteration, can be used not only when there is more than a single independent subproblem, but also with a single subproblem by choosing the k most attractive columns, for example.

The second aspect has to deal with the scheme adopted to select the nonbasic column to add to the master among a set of potentially attractive columns. The simplest scheme consists in selecting the first variable with a negative reduced cost. This scheme is not very efficient. It usually leads to many pivot operations. The pricing scheme that is traditionally used consists in selecting the variable with the most negative reduced cost. This rule was suggested by Dantzig. This scheme does not necessarily ensure the best improvement of the objective function, since the maximum value the new variable can reach is not taken into account. In order to improve the impact on the objective function, Gilmore and Gomory proposed to price out columns according to a "median" method. The items are divided in two groups, with low and high demands, respectively, and knapsack subproblems are solved with only the items of the second group. This allows the corresponding variable in the master to take a larger value. In fact, the most generic and effective alternative to the Dantzig's rule is the steepest edge scheme, which reduces considerably the number of iterations [47]. However, this is also the most expensive computationally. In [83], other pricing schemes are further described, including deepest cut, lagrangean pricing [82] and lambda pricing [17].

2.5 Column Generation and Branch-and-Bound

Combining column generation and LP branch-and-bound, with the aim of finding integer optimal solutions, is not straightforward. In fact, the exercise is as difficult as adding valid cutting planes to the restricted master, and keeping the subproblem computationally tractable. Almost twenty five years were elapsed before these two techniques began to be used in a common algorithm [32, 30]. Since then, there have been many attempts to optimize large scale integer programs using branch-and-price algorithms [92, 29, 120, 99, 119, 115, 4].

Column generation is necessary at all the nodes of the branch-and-bound search tree, since the columns enumerated in the restricted master at the root may not be enough to construct the optimal integer solution. Using column generation only at the root node, before any branching constraint has been enforced, leads necessarily to approaches that are heuristic. Furthermore, generating the complete set of columns, as in [57], is an approach which is admissible only for small size instances. Hence, branching

schemes must be devised so as to be compatible with the pricing subproblems.

A first remark is the fact that branching on the variables of the master must be avoided. What happens in these cases is simply the regeneration of columns that are already in the master, forcing the identification in the pricing subproblems of the second best column, or the third, or even more, depending on the depth of the node in the branch-and-bound tree.

Many results have been reported in the literature concerning branch-and-price algorithms applied to problems with binary variables [120, 124]. In [120], for example, the authors solve the binary cutting stock problem, which is characterized by ordered quantities that are equal to the unit. The master is a set partitioning problem, the pricing subproblem a knapsack problem, and branching is done on the variables of the former. Two branches are created whenever the solution of the master is fractional, and the continuous bound is strictly lower than the incumbent. No formal constraints are enforced in the master. In fact, the branching constraints are directly reported to the subproblem. In the left branch, the one associated with the "greater than or equal to" branching constraint, two of the items are replaced by a single item with the sum of their sizes. The subproblem loses one row, the subproblem remains a knapsack problem with an item less. On the other hand, the right branch is much harder, since the branching constraints impose a reformulation of the original knapsack subproblem. Concerning branch-and-price with general integer programs, a general framework is given in [125]. The authors show how to report to the subproblem the dual values related to the branching constraints, by using additional binary variables.

As happens with cutting planes, branching decisions should be taken on the original variables in order to keep the master problem compatible with the pricing subproblem at each node of the branch-and-bound tree. In practice, branching constraints on original variables partition the set of columns in the restricted master problem. Here, the compatibility between the branching scheme and the pricing subproblem is related to how difficult it is to know if a solution of the pricing subproblem belongs or not to a particular partition.

2.6 Cutting and Packing Problems: Overview

Cutting and packing are processes arising in the most diverse contexts. We find them for example in manufacturing industries, transportation and information technology. As a consequence, the subject has been treated by researchers from different fields, as management science, operations research, computer science and mathematics. A further symptom of the importance of the area, and the challenges it raises, is the research effort involved, which has been of considerable volume. Many original contributions are given in the literature, some of them gathered in special issues of scientific journals,

and the topic became a traditional reference in many textbooks for a long time [80, 21].

Cutting and packing problems are combinatorial optimization problems. As happens with many other problems of this kind, they are easy to state, and difficult to solve. The standard problem is defined as follows: given a set of small and large objects, how should the small objects be assigned to the large ones in order to optimize a given criterion. The typical restrictions to which a cutting or packing plan is submitted are the impossibility for the small objects to overlap, and the limited capacity or length of the large objects. Beyond this common definition, there exist many variations and extensions. Dimensionality, the shapes of the figures and orientation are some examples. The distinction between cutting and packing operations is already an evidence of the existence of activities that are different in practice. However, cutting and packing problems are often treated indistinctly. This is due to the fact that, from a theoretical standpoint, they are just two ways of looking at the same problem. While cutting problems are essentially based on the notion of material, packing is related to the notion of space.

The profusion of results reported in the literature [38], under different taxonomies, motivated the development of an unifying classification system by Dyckhoff [37], which allows an easier discussion and comparison of the approaches devised for cutting and packing problems. The typology suggested sets the frontiers beyond which the increase in the complexity of the solution approaches is clearly established. Dyckhoff proposed a classification scheme based on four characteristics, further divided in subtypes as follows

1. dimensionality

   (N) N-dimensional

2. kind of assignment

   (B) all large objects and a selection of small objects

   (V) a selection of large objects and all small objects

3. assortment of large objects

   (O) one large object

   (I) many identical large objects

   (D) different large objects

4. assortment of small objects

   (F) few small objects of different figures

   (M) many small objects of many different figures

(R) many small objects of relatively few different figures

(C) many identical small objects

Noting that this typology does not consider the kind of assortment treated by Gilmore and Gomory [53], *i.e.*, the case in which there are few groups of different stock lengths, Gradisar *et al.* [60] suggest adding the following subtype

(G) few groups of identical large objects.

This first drawback of the Dyckhoff's typology is further discussed in [127], where other weaknesses of this classification scheme are also described. Apart from this lack of homogeneity, with this classification scheme, the same cutting or packing problem can reasonably be assigned to different categories. To overcome these weaknesses, a new typology has been developed by Waescher *et al.* [127]. Their aim was to catch all the characteristics of cutting and packing problems in a classification scheme that can be widely accepted by researchers. As a way of testing it in practice, the authors classified almost 300 papers published in the last decade.

As we can see from the typologies proposed in the literature, one of the essential distinctions that is made among cutting and packing problems is related to the geometry of the objects that are involved. This characteristic affects deeply the complexity of the solution approaches, mainly because it conditions the degrees of freedom of the problems. As an example, in one-dimensional problems, nobody has to care about the items' orientation. Some constraints on the way the cutting or packing can be done are enforced frequently, which, in some way, simplify the approaches. Imposing a layered organization of the small items within the large objects is common. This arises with two-dimensional problems, as in the so-called Level Packing Problem [81], for example, and with three-dimensional problems [16]. Gilmore and Gomory [54] already noted that, in practice, cutting problems with more than one dimension are treated in more than a single stage. For the two-dimensional case, two series of guillotine cuts are usually applied on the large objects, producing a set of strips that are subsequently cut varying by 90^o the angle of the guillotine. Another property that brings even more difficulties is the regularity of the shapes of the small items. Problems of this type, which are frequent in the leather and textile industries, have been classified under the designation of "Nesting Problems" [56, 18], among others.

Another key distinction between cutting and packing problems is the general objective pursued. There are two trends here. The first consists in determining a solution in order to maximize a global profit directly related to the small items chosen. The alternative is to obey to supply requirements, and to optimize certain operational costs. These costs may be related to the waste incurred, or to the trim loss in the case of cutting processes. As an example of the first type of problems, we can mention the well known Knapsack Problem [88], with a single dimension, in which a set of small

items with an associated profit and size have to be chosen such that the capacity of the knapsack is not exceeded. The standard Cutting Stock Problem belongs to the second category.

Many extensions to the standard versions of cutting and packing problems are allowed, some of them having deserved the attention of researchers. The most obvious is perhaps the availability of different types of large objects. Differences may arise simply in the length or capacity of these objects, but they can also involve measures of quality. Another extension is the existence of additional constraints in the cutting and packing plans, together with the traditional capacity constraints [102]. For example, the number of pieces that can be cut from a stock roll may be limited to the available number of cutting knives. Another trend in cutting and packing research has been to take into account the auxiliary problems that may arise. In cutting environments, for example, maximizing the similarities among the patterns may help to accelerate operations, and eventually to reduce the waste related to the positioning of the knives. The related optimization problem is called the "Setup Minimization Problem", or, alternatively, the "Pattern Minimization Problem". We explore it further in Chapter 5.

In this thesis, we are concerned with one-dimensional cutting and packing problems. In the sequel, we will concentrate on these specific problems, and briefly recall the models and algorithms that have been proposed in the literature for the standard case.

2.7 One-Dimensional Cutting and Packing

2.7.1 Models

The standard cutting and packing problems in a single dimension, and whose aim is to minimize the consumption of resources, are usually referred to as the Cutting Stock Problem and Bin-Packing Problem, respectively. The linear programming models that have been proposed for them so far can be divided in four categories: the assignment formulations, the pattern-oriented formulations, the one-cut formulations and the flow models. We have already seen the assignment formulation, in Section 2.1.2. Historically, the model from Gilmore and Gomory [52] followed this assignment formulation due to Kantorovich [76]. Instead of using a clear reference to each roll or bin, their model is based on the enumeration of the feasible combinations of items within the large objects. It is obtained applying a Dantzig-Wolfe decomposition, in which only the demand constraints (3.2) are kept in the master problem, and it states as follows

$$\min \quad \sum_{p \in P} \lambda_p \tag{2.6}$$

$$\text{subject to} \quad \sum_{p \in P} a_{ip} \lambda_p \geq b_i, \; i = 1, \ldots, m, \tag{2.7}$$

$$\lambda_p \geq 0 \text{ and integer, } p \in P. \tag{2.8}$$

Each column p in the set P represents a cutting or packing pattern, with a_{ip} representing the number of items of size w_i that are in this pattern. What is common to all the patterns in P is the fact that the sum of the corresponding item sizes is less than or equal to W, the length or capacity of the large object (roll or bin). This column generation reformulation (this designation is acceptable in practice since a complete enumeration of the columns is clearly out of question) is better than the assignment model in critical aspects. First, there is no symmetry in its solution space, as mentioned in a previous section. Furthermore, its linear programming bound is rather strong. For most of the instances, the integrality gap is smaller than the unit [85]. Some of them may have a gap greater than 1, as it is shown in [86], but it seems that it never exceeds 2 [100]. The major drawback of this model comes from its size, which can be tackled by generating the patterns dynamically.

In the model (2.6)-(2.8), each column is related to an operation (cutting or packing) which involves a whole large object. In the one-cut models, which have been proposed for the Cutting Stock Problem in [36, 109], the principle is to determine how to apply a single cut (a one-cut) on an original or residual piece of material. A one-cut divides the raw material in two pieces: an ordered item and a residual object. The latter can be trim loss, a portion to be cut further, or another ordered width. The model proposed by Dyckhoff considers the case in which different stock lengths are available. In the sequel, D represents the set of item widths to cut from the stock rolls whose lengths belong to $S = \{W_1, \ldots, W_K\}$. The set of residual objects whose length is sufficiently large to cut an item is denoted by R. The decision variables $y_{p,q}$ indicate the number of times a piece with length p is to be cut so as to produce an item of width q, and a residual object of width $p-q$. The z_k variables indicate the number of stock rolls with lengths W_k that are used. The Dyckhoff's model is presented next.

$$\min \sum_{k=1}^{K} W_k z_k \tag{2.9}$$

subject to

$$z_k + \sum_{p \in D: p+q \in S \cup R} y_{p+q,p} \geq \sum_{p \in D: p<q} y_{q,p}, \ \forall q \in S, \tag{2.10}$$

$$\sum_{p \in S \cup R: p>q} y_{p,q} + \sum_{p \in D: p+q \in S \cup R} y_{p+q,p} \geq \sum_{p \in D: p<q} y_{q,p} + N_q, \ \forall q \in (D \cup R) \setminus S, \tag{2.11}$$

$$y_{p,q} \geq 0 \text{ and integer, } p \in S \cup R, \ q \in D, \ q < p, \tag{2.12}$$

$$z_k \geq 0 \text{ and integer, } k = 1, \ldots, K. \tag{2.13}$$

Inequalities (2.10) are the definition constraints for the variables z_k. In (2.11), the value N_q stands for the demand of q, when this is an ordered item, or 0 if this is a residual

piece of material. To cut an ordered item from a roll of length W_k, we must use one of the stock rolls available, or cut any other piece of material so as to produce a roll with this length. This constraint is formulated as (2.10). The satisfaction of the demand for an item of width w_i, and/or the satisfaction of the needs for residual objects with the same width, is guaranteed by cutting explicitly this width from a larger piece, or implicitly by generating a residual piece of material with this desired width (constraints (2.11)). An advantage of this formulation is that the number of variables is not as big as in the model of Gilmore and Gomory, but it is still pseudo-polynomial as the number of constraints. Furthermore, it has symmetry.

The last models are based on flow variables. The first model that we recall here, and which is due to Valério de Carvalho [114, 115], represents each integer position within the large object as a node in a graph, say G, and an item placed at a certain distance from the left border as an arc between the two nodes associated to the initial and final position of this item. That explains the designation "position indexed models" used in [116]. For an arc set denoted by A, and arc flow variables x_{ij} between two nodes i and j belonging to A, we can formulate this model as follows

$$\text{min. } z \tag{2.14}$$

subject to

$$-\sum_{(i,j)\in A} x_{ij} + \sum_{(j,l)\in A} x_{jl} = \begin{cases} z, & \text{if } j = 0, \\ 0, & \text{if } j = 1, \ldots, W-1, \\ -z, & \text{if } j = W, \end{cases} \tag{2.15}$$

$$\sum_{(i,i+w_d)\in A} x_{i,i+w_d} \geq b_d, \ d = 1, \ldots, m, \tag{2.16}$$

$$x_{ij} \geq 0 \text{ and integer}, \ \forall (i,j) \in A. \tag{2.17}$$

This model is equivalent to the one of Gilmore and Gomory [115], and, hence, the continuous bounds are the same. As in the one-cut models, it is not completely free of symmetry, even if some restrictions on the arc set A can considerably reduce it. The number of constraints, which is pseudo-polynomial, is its major weakness. In [115], the author proposes to generate the flow conservation constraints only as needed.

The second flow model, suggested in [28] for the Bin-Packing Problem, consists in an analogy with vehicle routing problems. A decision whether to pick an object or not, and place it in a vehicle with a limited capacity, models each packing operation. Here, the vehicles are the large objects, and the clients and their corresponding loads represent the small items. Initially, the vehicles are empty. Their tours begin in a depot represented by a node o, and end at the same depot, represented by a different node d. The items are represented by a node, belonging to a set N. The set of arcs,

representing valid sequences of items, is denoted by A. There are two types of variables: the binary x_{ij}^k indicate whether the vehicle k, $k = 1, \ldots, K$, uses arc (i, j) or not, *i.e.*, whether there is, in the corresponding bin, an item of size w_j that follows another item of size w_i; the W_i^k variables represent the capacity used by vehicle k when it leaves the node i. The resulting model is nonlinear. For the case of a homogeneous fleet of vehicles (identical bins), this formulation states as follows

$$\text{min.} \sum_{k \in K} \sum_{(o,j) \in A^k} x_{oj}^k \tag{2.18}$$

subject to

$$\sum_{k \in K} \sum_{j:(i,j) \in A^k} x_{ij}^k = 1, \ \forall i \in N, \tag{2.19}$$

$$\sum_{(i,j) \in A^k} x_{ij}^k - \sum_{(i,j) \in A^k} x_{ji}^k = 0, \ \forall i \in N^k, \ k = 1, \ldots, K, \tag{2.20}$$

$$x_{ij}^k (W_i^k + w_j - W_j^k) \leq 0, \ \forall (i, j) \in A^k, \ k = 1, \ldots, K, \tag{2.21}$$

$$w_i \leq W_i^k \leq W, \ \forall i \in V^k, \ k = 1, \ldots, K, \tag{2.22}$$

$$x_{ij}^k \in \{0, 1\}, \ \forall (i, j) \in A^k, \ k = 1, \ldots, K. \tag{2.23}$$

As noted in [116], the nonlinearities can easily be dealt with, by applying a Dantzig-Wolfe decomposition in which only (2.19) is kept in the master problem.

2.7.2 Algorithms

In [23], Coffman *et al.* give a comprehensive survey on the various approximation algorithms that have been proposed for the Bin-Packing Problem, and which are applicable to the Cutting Stock Problem as well. The authors explore the worst-case and average behaviors of the different algorithms. In [39, 40], other approaches are studied which are based on the theory of evolutionary algorithms. The considerable investment on heuristic approaches is motivated by the fact that both problems are well known to be NP-hard.

The MTP algorithm of Martello and Toth [88] is among the exact approaches proposed for the Bin-Packing Problem. Their approach is enumerative, based on branch-and-bound, and combines lower bounding strategies with heuristic methods at each node. Scholl *et al.* [104] report better results using a branch-and-bound procedure with a new branching scheme, reduction procedures, lower bounds, and heuristics, as a tabu search procedure, among others. As we can see, the fast computation of strong lower bounds is important in these approaches. Schwerin and Waescher [106], and later Fekete and Schepers [42], proposed alternative lower bounds for the Bin-Packing Problem.

Linear programming based algorithms have achieved an appreciable success in solving exactly the cutting stock and bin-packing problems. The pioneers were Gilmore and Gomory [52, 53], who combined column generation with rounding to obtain near optimal solutions. More recent attempts resort to cutting planes [101], or to combining column generation with branch-and-bound [120, 122, 114, 115, 27, 26].

Chapter 3

The Multiple Length Cutting Stock Problem

Many heuristic approaches have been devised and described throughout the literature for the multiple length cutting stock problem. On the opposite, there have been only a few results reported regarding attempts to exactly solve this problem. In fact, results for exact solution algorithms appeared only very recently. In the sequel, we investigate the efficiency of a branch-and-price-and-cut algorithm, developed using two equivalent models. The whole procedure consists on the execution of column generation at each node of a branch-and-bound tree, and a further strengthening process based on two families of valid cutting planes. To measure its efficiency, we tested our algorithm using different instances from the literature.
In practical applications, when many stock lengths are available, a common problem that arises is the one of selecting the subset of lengths that will be used. This problem is known as the assortment problem. Here, we consider the simultaneous optimization of the cutting plan and selection of the stock lengths. To designate the resulting problem, we use the name "combined assortment and trim loss minimization problem". The branch-and-price-and-cut algorithm developed was extended to this problem. Many computational results are reported for a set of randomly generated instances.

Keywords: Multiple Length Cutting Stock Problem, Column Generation, Cutting Planes, Branch-and-Price-and-Cut, Combined Assortment and Trim Loss Minimization Problem

3.1 Introduction

In this chapter, we are concerned with a generalization of the thoroughly studied standard cutting stock problem, in which multiple stock lengths are available instead of a single one. This is a natural extension, but not the only one which has deserved the attention of the research community. Indeed, further distinctions can be made among the rolls. There is for example reported research regarding problems where the rolls have different quality grades [111]. In these problems, the feasibility of a cutting pattern is not only conditioned by the length of the roll, but also by the quality requirements of the orders. These are problems that arise frequently in the paper, wood and steel industries [97]. For different stock rolls, there may be also different production or distribution centers, in different locations. Haessler and Sweeney gave a formulation for a related problem, namely the cutting stock problem with multiple lengths and freight costs [63].

With different stock lengths, and consequently a greater choice of feasible cutting patterns, one can naturally expect to obtain better solutions in terms of material usage than one would get with only one stock length. This intuitive statement is defended by Gilmore and Gomory in one of their pioneering papers [53]. The authors gave computational evidence that supports this idea. However, finding such improved solutions requires solving an optimization problem with a somewhat harder cost function. The integer round-up property, formally described by Marcotte [85], no longer applies to the cutting stock problem with multiple stock lengths, in which the cost of a roll is proportional to its length.

Additional constraints may also be considered. Rolls' availability may be limited for example. Sometimes, this is the principal motivation for using different stock lengths. The maximum number of rolls that can be used may also be bounded. In this case, the decision problem is twofold. The problem of selecting the best set of cutting patterns arises only after one has decided which stock lengths to use [64]. This problem is referred to as the combined assortment and trim loss minimization problem.

The Multiple Length Cutting Stock Problem (MLCSP) is NP-hard [49]. Unless P=NP, no absolute approximation scheme can be devised that solves it in fully polynomial time. As a consequence, most of the research effort has been initially devoted to the development of heuristics and approximation algorithms for which only worst-case performance could be guaranteed. In [53], Gilmore and Gomory applied their column generation algorithm to a problem called the Machine Balance Problem, and studied the effect of stock lengths. Roodman [98] found near optimal integer solutions using a procedure based on the solution of the LP relaxation, complemented by heuristically generated columns. Recently, Holthaus [67] presented a heuristic method in which column generation was combined with rounding and fixing algorithms and other methods

to solve the residual problem. Additional heuristic procedures can be found in [58, 59].

Other publications address the Variable Sized Bin-Packing Problem (VSBP), which is, in the bin-packing literature, the counterpart of the MLCSP. The difference between the VSBP and the MLCSP is essentially in the levels of demand. Friesen and Langston [48] present three approximation algorithms, with asymptotic worst-case performance bounds of 2, $\frac{3}{2}$ and $\frac{4}{3}$, respectively. Murgolo [91] describes a fully polynomial asymptotic approximation scheme, and, more recently, Chu and La [20] derive four approximation algorithms with absolute worst-case performance of 2, 2, 3 and 2+ln 2, respectively.

There are also some interesting results regarding the solvability of problems with special properties. In a recent paper, Kang and Park [75] present two greedy algorithms that solve to optimality the VSBP with divisible item and bin sizes. Their approach is based on the well-known First-Fit and Best-Fit Decreasing algorithms. The bin types are chosen one by one, and the items are packed using the FFD or BFD scheme, starting from the largest and repacking iteratively the items to the second largest bin, and so on. The resulting heuristics are designated by Iterative First-Fit Decreasing and Iterative Best-Fit Decreasing. For the case in which only the bin sizes are divisible, the authors show that the solution given by their algorithms is never worse than $\frac{11}{9}z^* + \frac{44}{9}$, for an optimal solution of value z^*. For the general case, the algorithms give a solution which is never greater than $\frac{3}{2}z^*+1$. Furthermore, the authors present a counterexample that refutes a prior conjecture from Coffman *et al.* [22], stating that a modified version of the FFD algorithm could give optimal solutions to the VSBP in which the item sizes are divisible and the bin sizes are multiples of all the item sizes.

Only recently, results have been reported on attempts to develop computationally practical exact algorithms. In his PhD thesis [90], Monaci addresses the VSBP. He describes various lower bounding procedures, some of them trivial and others more sophisticated, and presents three heuristics, namely a First-Fit type algorithm, a greedy algorithm, and an improved version of this greedy procedure, called the diving algorithm. The author also developed a local search routine to improve iteratively a given solution. Exact solutions are found using a branch-and-bound algorithm with a simple branching rule proposed by Martello and Toth for the bin-packing problem [88]. Node selection is done following a best-bound first strategy. In each branching node, an item is assigned to either one of the opened bins or to a new one, and heuristic solutions are computed using the greedy and diving algorithm. Local search is finally applied to improve these solutions. The branching nodes are given a penalty value that depends on how far these final solutions are from the compulsive assignment of items to bins given by the branching scheme. Computational experiments are reported for a set of 300 instances, with a maximum of 5 different rolls and 500 items. The algorithm was able to solve 78% of the instances within a time limit of 15 minutes for each.

Another exact approach was proposed recently by Belov and Scheithauer, in [13].

The authors combine Chvátal-Gomory cutting planes with column generation, bringing to the latter some added complexity. In order to price a new column correctly, the coefficients in the enumerated cuts must be anticipated. This leads to a pricing subproblem that becomes a general IP problem without any special structure, much harder than a knapsack problem, and which has to be solved by branch-and-bound. The authors also give a rounding heuristic, based on the sequential value correction method, which seems to be essential to the success of their overall algorithm. Their paper reports on an extensive set of computational experiments.

In this chapter, we explore an exact branch-and-price-and-cut algorithm for the MLCSP relying on two formulations described in the literature, and show through comparative experiments that it is possible to get better results than those published recently with an algorithm that does not rely on any sophisticated rounding scheme. As a matter of fact, our algorithm was able to solve all the instances tested by Monaci, and to improve the results obtained on some instances used in [13]. In the sequel, we present the main features of this algorithm, like the branching scheme, the cutting planes and the ways early node termination is performed. A prior remark is that the pricing subproblem is not drastically modified by the branching scheme adopted, and can still be solved in pseudo-polynomial time. In practice, it is solved very efficiently.

Column generation is at the heart of our solution approach for the MLCSP, and many others referred to above. The model it solves, which results from a Dantzig-Wolfe decomposition of a compact formulation [119, 116], is stronger, and leads to improved lower bounds. However, it is well known that column generation procedures suffer from slow convergence induced by undesirable behaviors such as the primal degeneracy and the excessive oscillations of the dual variables. We do not approach the topic of stabilized column generation in this chapter. The next chapter will be fully devoted to it instead.

A final remark concerning the extensive research that is reported for the online version of the VSBP has to be done. This problem arises when the items arrive one by one, and the decision about where to pack them must be taken immediately. In a recent paper, Seiden *et al.* [108] give lower and upper bounds for the problem, along with solution algorithms. In [130], Zhang analyzes the worst-case performance of the First-Fit Decreasing scheme applied to the online VSBP. Other references to this problem may also be found in [24, 79, 107].

This chapter is organized as follows. We first review the IP formulations for the MLCSP, and describe next the details of the branch-and-bound procedure. The cutting planes which were used to strengthen the model are then described. The results of our computational experiments are reported afterwards. The second part of this chapter is devoted to the combined assortment and trim loss minimization problem. The algorithm previously introduced is extended, and its behavior analyzed with different

computational experiments.

3.2 Problem Formulations

3.2.1 A Column Generation Model

In the Multiple Length Cutting Stock Problem, we are given K types of stock rolls with integer lengths W_k, such that $W_k \neq W_{k'}$, for all $k \neq k'$, and m sets of items with sizes w_i, $i = 1, \ldots, m$. We consider the case in which the availability of a stock roll of the k^{th} class is limited to B_k units. The item demands are denoted by b_i, and may be greater than the unit. Throughout this chapter, we will assume that the items and the rolls are sorted in order of decreasing sizes and lengths, respectively. The objective analyzed here is to find the cutting plan which minimizes the sum of the roll lengths, or equivalently, the trim loss. Note that, due to the availability constraints on the stock rolls, a particular instance of the problem may not have a feasible solution.

The formulation presented next is an extension of the well known model for the standard Cutting Stock Problem proposed by Gilmore and Gomory. This model was used by the authors to formulate their Machine Balance Problem [53]. For any moderate size instance, this model has a huge set of columns, or decision variables. This forbids solutions based on complete enumerations, such as the one proposed by Goulimis, in [57]. The model used instead relies on a restricted set of columns, which is completed dynamically by other columns that may be useful. This results in a so called column generation model, which can be stated as follows

$$\min \quad \sum_{k=1}^{K} \sum_{p \in P^k} W_k \lambda_p^k \tag{3.1}$$

$$\text{subject to} \quad \sum_{k=1}^{K} \sum_{p \in P^k} a_{ip}^k \lambda_p^k \geq b_i, \ i = 1, \ldots, m, \tag{3.2}$$

$$\sum_{p \in P^k} \lambda_p^k \leq B_k, \ k = 1, \ldots, K, \tag{3.3}$$

$$\lambda_p^k \geq 0 \text{ and integer}, \ k = 1, \ldots, K, \ p \in P^k. \tag{3.4}$$

For a stock roll of length W_k, there is a set of feasible cutting patterns, which is denoted by P^k. In formulation (3.1)-(3.4), each column represents a feasible cutting pattern for one of the stock lengths. It consists in a vector $(a_{1p}^k, a_{2p}^k, \ldots, a_{mp}^k; \ldots, 1, \ldots)^T$, in which a_{ip}^k denotes the number of items i cut from a roll of length W_k. The cutting pattern usages are the only decision variables, and are denoted by λ_p^k.

Constraints (3.2) ensure the satisfaction of the demands for each item, whereas the rolls' availability constraints are represented by (3.3). Instead of "greater than or equal to" inequalities in (3.2), one might prefer equality constraints since we consider here the simple case where demand is to be met exactly and overproduction is discarded as waste. However, as shown in [52], it is always possible to recover a solution of equal cost in which demand is met exactly, *i.e.*, with the slack variables in (3.2) equal to 0. On the other hand, with equality constraints, the dual problem has variables that are not restricted in sign. When inequalities are used, part of this dual space is cut, leading to column generation algorithms that converge faster. Restricting the dual solution space is a nice feature as will become clear in the next chapter, and therefore we will keep our demand constraints defined as inequalities.

The previous model can be obtained applying a Dantzig-Wolfe decomposition to an extended version of the Kantorovich model [76] for the MLCSP, or alternatively to the flow model described in [116]. Both are compact or original models [126]. The former relies on assignment variables and is known for its poor LP bound and symmetry. The flow model gives an LP lower bound which is equivalent to the one obtained with (3.1)-(3.4). However, the set of flow conservation constraints may be quite large, and even when these constraints are only enumerated as needed, they may still be a burden for any solution algorithm. This makes the column generation model (3.1)-(3.4) one of the most referenced to and used models for the CSP, and also for the MLCSP. In this chapter, we take advantage of both the column generation and flow models. In the following section, we briefly describe the latter.

3.2.2 A Compact Flow Model

In [115], an alternative compact model with flow variables was described for the CSP. As happens with the Gilmore and Gomory model, its extension to the MLCSP case is straightforward [116]. The model is defined over a graph with as many vertices as the length of the largest roll plus one. Each vertex represents a discrete position within the roll, and each arc the placement of an item in a precise area. A cutting pattern consists in an uninterrupted concatenation of arcs starting at vertex 0, the left border of the roll, and ending in vertex W_k. The flow over this sequence of arcs is the respective pattern usage.

Formally, let $G = (V, A)$ be the graph, with V the set of vertices and A the set of arcs. Within the arc set, there are item and loss arcs. For an item arc, say (i, j), we have $(i, j) \in A$, if $0 \leq i < j \leq W_1$, and $j - i = w_d$, for any $1 \leq d \leq m$. Remember that W_1 is the size of the largest roll. The loss arcs are unit length arcs representing the unused portions of the rolls. The graph G has $O(mW_1)$ arcs, and exactly $W_1 + 1$ vertices.

A pattern is defined as a set of items that are cut in a roll of a certain length. Given

a set of items, there can be many different ways of placing them. As a consequence, without any further restrictions to A, the flow model may be highly symmetric. Different reduction criteria can be applied to reduce this symmetry, as those proposed in [115]. Loss arcs, for example, are left for the end of the roll. An ordering of the arcs according to their lengths is also recommended. The following example illustrates the flow model for a simple instance of the MLCSP.

Example 3.1 Consider an instance with a set of rolls of lengths 7, 5 and 4, and items of sizes 3, 3, 3, 2 and 2. The rolls' availability are 1, 1 and 3, for each length, respectively. The total size of the ordered items is $\sum_d w_d b_d = 13$, for a total roll length of $\sum_k W_k B_k = 24$. The complete graph G is depicted in Figure 3.1. The set of arcs is the one obtained after applying the reduction criteria referred to above. □

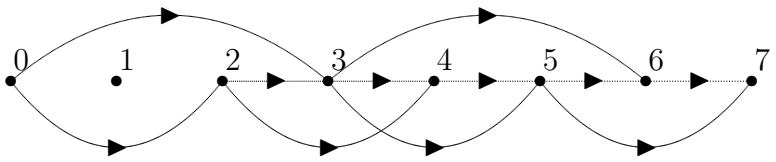

Figure 3.1: Graph G for the instance of Example 3.1

The corresponding IP formulation is as follows.

$$\text{min.} \sum_{k=1}^{K} W_k z_k \tag{3.5}$$

subject to

$$-\sum_{(i,j)\in A} x_{ij} + \sum_{(j,l)\in A} x_{jl} = \begin{cases} \sum_{k=1}^{K} z_k, & \text{if } j = 0, \\ -z_k, & \text{for } j = W_k,\ k = 1,\ldots,K, \\ 0, & \text{otherwise,} \end{cases} \tag{3.6}$$

$$\sum_{(i,i+w_d)\in A} x_{i,i+w_d} \geq b_d,\ d = 1,\ldots,m \tag{3.7}$$

$$z_k \leq B_k,\ k = 1,\ldots,K, \tag{3.8}$$

$$x_{ij} \geq 0 \text{ and integer},\ \forall (i,j) \in A, \tag{3.9}$$

$$z_k \geq 0 \text{ and integer},\ k = 1,\ldots,K. \tag{3.10}$$

The MLCSP is formulated as a minimum weighted flow problem in G, with additional constraints for the item demands (3.7) and rolls' availability (3.8). Flow conservation is imposed through the constraints (3.6). The flow variables are denoted by x_{ij}, whereas z_k denotes the number of rolls with length W_k used.

According to the flow decomposition principle (see [1]), a set of arc flows in G can be decomposed into a set of path and cycle flows. Cycles in G may be defined by considering K additional feedback arcs starting at positions W_k, $k = 1,\ldots,K$, and ending at vertex 0.

Formulation (3.5)-(3.8) has $O(W_1 + m + K)$ constraints, a number which depends on the length of the largest roll. As referred to in [116], its main value comes from its compatibility with branch-and-price frameworks. Branching schemes relying on this model can be devised without inducing any important change to the structure of the original pricing subproblems.

3.3 Branch-and-Price

In this section, we review the main features of the branch-and-price part of the overall algorithm investigated here. We begin by describing the elements of the LP problem solved to obtain the continuous bounds at the nodes of the branching tree. We proceed with the description of the branching scheme, the presentation of the bounds computed along the solution process, and a reference to the simple rounding procedure used.

3.3.1 LP Relaxation

The problem solved at the nodes of the branch-and-bound tree is the LP relaxation of (3.1)-(3.4). Given the dimension of this model, it is more efficient to use it to compute the LP bound rather than the flow model. The number of constraints of a linear program is indeed an important parameter that may condition the efficiency of computational methods. In [10], the average empirical complexity of the simplex method was estimated to be $O(m^2 n)$, being m and n the number of constraints and decision variables, respectively. The increase is quadratic in the number of constraints.

Being a complete enumeration of the columns impracticable, we resort to a partial enumeration of the pattern set, and deal with a so called restricted master problem. A solution to the latter is an upper bound on the LP optimum. After the master is optimized, K pricing subproblems are solved to identify those columns with a negative reduced cost that may enter the basis and provide a better cost solution. At the root node of the branch-and-bound tree, we solve K pricing subproblems per iteration of the column generation procedure using the MT1 algorithm from Martello and Toth [88]. In the remaining nodes, we use dynamic programming because of the specific branching constraints that are enforced. In this case, a single run of the algorithm is enough to solve these K subproblems. Only the most attractive pattern for each stock length is added to the master. When there are no more attractive columns, column generation stops. We do not consider any scheme for removing the columns in the master. Once they are added, they are not removed anymore.

The first restricted master problem is initialized with an artificial column that ensures its feasibility, whatever the branching node. Infeasibility may occur at the beginning if the initialization heuristic fails to find a valid solution, or after a branching

constraint or a cutting plane has been enforced. This column has coefficients equal to the right hand side of the "greater than or equal to" constraints, and zeroes for the other types of constraints. The problem may be infeasible only for the restricted set of enumerated columns, or for the whole set of patterns. In the first case, the artificial variable will begin with a positive value and attractive columns will certainly be found. In the last case, the artificial variable will remain positive, and the cost of the solution must be chosen such that the corresponding node can be pruned by bound. In fact, the cost of the artificial column must be at least equal to the incumbent.

Besides the artificial column, the LP master is initialized with a set of patterns given by an extension of the First Fit Decreasing packing heuristic. The rolls are chosen (opened) in order of decreasing lengths, and the items are placed, also in order of decreasing sizes, in the first opened roll into which they fit, or, if there is no space in any of them, in a new one. Rolls' availabilities are taken into account. As a consequence, the heuristic may fail to find a solution.

Only minimal (or complete [110]) patterns are considered. For these patterns, the space that is unfulfilled is always smaller than the size of the smallest item. This is a true restriction on the set of permissible patterns, that forbids the use of equality demand constraints.

3.3.2 Branching Strategy

In [123], Vanderbeck suggests that branching should be done on columns sharing an identical property that can easily be identified in the pricing subproblem. A possible way of defining a partition of the columns is to select them according to the precise items that are in the corresponding patterns, or alternatively according to the position of an item within the roll, if an ordering of the items is previously assumed. The latter partitioning rule leads to a branching scheme on the flow variables of formulation (3.5)-(3.10). We will use it, since it does not induce any important modification to the original knapsack subproblems. After solving the LP relaxation of (3.1)-(3.4), a fractional solution can be converted into a set of flows, an arc or a set of arcs with fractional flows identified, and a branching constraint enforced back in the Gilmore and Gomory model.

When branching on the flow variables, many different schemes may be devised. Ideally, a branching scheme should be such that the resulting branch-and-bound search tree is balanced and free of symmetry. In a balanced tree, the set of feasible solutions in a parent node is clearly different from the feasible region of all its descendants. On the other hand, with an unbalanced tree, one of the branching nodes has almost the same set of solutions as its parent node. In these circumstances, we can not expect to progress a lot. Symmetry is another major cause of ineffectiveness of branch-and-bound algorithms. There is symmetry when different assignment of values to variables

lead to solutions that are identical in practice. Branching, whose principal aim is to exclude a given fractional solution, does not ensure that this solution will appear again in a deeper node. Our branching scheme is based on a selection of the columns of the model of Gilmore and Gomory, which is made by converting the continuous solution at a node into an equivalent solution of the arc-flow model, using a pre-defined ordering among the items. As we already mentioned, the LP model of Gilmore and Gomory has no symmetry.

To ensure a reasonably balanced tree, we developed a branching scheme with two levels. On the first level, we branch on fractional z_k variables of model (3.5)-(3.10), choosing the one that corresponds to the largest roll. Since there is a subproblem for each stock length, taking into account the dual variables for the resulting branching constraints is straightforward. If none of these variables is fractional, the solution may be still non integer. In this case, branching is done on the second level, on the item arcs. Among the fractional ones, we choose the leftmost arc, and break ties by selecting the arc that corresponds to the largest item. When the pricing subproblems are solved with dynamic programming, we can infer the position of an item within the knapsack, directly from the usual states which register the capacity used. Hence, we can easily apply the dual values of the branching constraints on the item arcs. However, this branching scheme prevents us from using other algorithms along the branch-and-bound tree, which may be computationally more efficient, such as the MT1 algorithm from Martello and Toth [88].

Another point of the branching scheme that has to be defined is the search strategy, *i.e.*, the order in which the nodes are selected. We know that branch-and-bound ends when the global upper and lower bounds are equal. In an algorithm, we can choose one of two strategies: we may privilege a fast improvement of the global lower bound, or alternatively we may direct the search so as to find an improved feasible integer solution as quickly as possible. The only way of improving the global lower bound is to strengthen the weakest LP relaxation. Hence, in this case, the node to select is the one with the lowest LP bound. This strategy is known as the best bound or best first search. To improve the incumbent, the best is to "dive" in the tree, and use the so called depth first search strategy. Alternatively, the two strategies can be combined in a hybrid search. The computational experiments that we have conducted indicate that the depth first search strategy provides very satisfactory results.

3.3.3 Lower Bounding

Its name makes it very clear: branch-and-bound is an optimization method that relies essentially on bounding techniques. With this principle in mind, besides the usual node fathoming that is made by comparing the LP optimum with the incumbent, the upper bound, we can avoid some unnecessary computations by comparing the LP values, or

other bounds described in the literature, with the global lower bound.

Sometimes, one can stop the column generation procedure at a certain branch-and-bound node before the LP optimum is reached. Let z_{LB} be the global integer lower bound, and z^w the optimal LP value obtained at a node w for a specific set of enumerated columns. If z^w is less than z_{LB}, the node cannot obviously be fathomed, but the column generation procedure can be stopped, since $z_{LB} - 1$ is clearly unreachable.

A lower bound that is easy to compute, and traditionally used, is the Farley's bound [41], which gives a lower bound on the optimum of the LP master. It is used to fathom a node without having to solve the master to optimality. Let z_F^w be the best, the greatest, Farley's bound computed at a node w. If z_F^w is greater than or equal to the incumbent, the node will never produce any improving solution, and therefore it can be fathomed. This bound is computed using the LP value of the master and the reduced cost of the last most attractive column. Another fast way to compute a lower bound is the one due to Lasdon [80], which also relies on the columns with the most negative reduced costs and on the value of the master.

3.3.4 Rounding Procedure

After an optimal LP solution is available, some simple rounding operations may help in finding an integer solution better than the incumbent. Hence, we devised a rounding scheme that is used in each branching node when column generation stops, and the separation algorithms to be described later does not identify any violated cutting plane.

Taking into account the restriction on the rolls' availability, the positive variables that have a fractional part greater than a parameterized value are rounded up (0.5, in our implementation). The other variables are rounded down. After this step, there may be items whose supply exceeds the demand. Iteratively, we choose a pattern with excess items, remove the items that are in excess, and we try to transfer the remaining items to a smaller roll. Since there may have more than a single pattern with excess items, we select the one that leads to the greatest saving. This step ends when there are no more items in excess, or when we are not able to transfer a pattern to a smaller roll anymore. Each pattern that is not complete is treated as an opened roll. The heuristic proceeds by assigning the remaining items in a FFD manner. Experiments have shown that, although this is a very basic scheme, improved incumbents are frequently found.

3.4 Cutting Planes

3.4.1 The Level Cut

A solution to the MLCSP consists in an optimal combination of stock rolls with integer lengths. With a continuous bound derived from the LP relaxation of (3.1)-(3.4) that we

will denote by z^w_{LP} for a branching node w, we can hence compute a possibly stronger integer lower bound by finding the first integer combination of stock lengths greater than or equal to z^w_{LP}. Let this bound be denoted by z^w_{IP}. Its exact value is the solution of the following problem

$$z^w_{IP} = \min \quad \sum_{k=1}^{K} W_k y_k \tag{3.11}$$

$$\text{subject to} \quad \sum_{k=1}^{K} W_k y_k \geq \lceil z^w_{LP} \rceil , \tag{3.12}$$

$$y_k \leq B_k, \ k = 1, \ldots, K, \tag{3.13}$$

$$y_k \geq 0 \text{ and integer}, \ k = 1, \ldots, K. \tag{3.14}$$

The MLCSP has not the integer round-up property, and z^w_{IP} may indeed be greater than $\lceil z^w_{LP} \rceil$.

From this point forward, we will call "level cut" to the corresponding inequality enforced in the master. There are two different ways of introducing it in formulation (3.1)-(3.4). The first is by stating that the trim loss has to appear somewhere in the cutting plan:

$$\sum_{k=1}^{K} \sum_{p \in P_k} \left(W_k - \sum_{i=1}^{m} w_i a^k_{ip} \right) \lambda^k_p \geq z^w_{IP} - \sum_{i=1}^{m} w_i b_i.$$

Alternatively, we can set the bound on the total stock lengths used as follows

$$\sum_{k=1}^{K} \sum_{p \in P_k} W_k \lambda^k_p \geq z^w_{IP}.$$

Instead of solving problem (3.11)-(3.14), we compute z^w_{IP} using a reformulation into a model with binary variables of the knapsack problems' family as described in [88]. Let first $K' = \sum_{k=1}^{K} B_k$, y'_k, $k = 1, \ldots, K'$, be the decision variables of model (3.11)-(3.14) expressed with binary variables instead of general integer ones, and $\bar{y}'_k$ be the complement of the latter, *i.e.* $\bar{y}'_k = 1 - y'_k$. The bound z^w_{IP} can then be obtained as follows

$$\begin{aligned}
z^w_{IP} &= \text{min.} \left\{ \sum_{k=1}^{K'} W_k y'_k : \sum_{k} W_k y'_k \geq \lceil z^w_{LP} \rceil, \ y' \in \mathbb{B}^{K'} \right\} = \\
&= \text{min.} \left\{ \sum_{k=1}^{K'} W_k (1 - \bar{y}'_k) : \sum_{k=1}^{K'} W_k (1 - \bar{y}'_k) \geq \lceil z^w_{LP} \rceil, \ \bar{y}' \in \mathbb{B}^{K'} \right\} = \\
&= \text{max.} \left\{ \sum_{k=1}^{K'} W_k \bar{y}'_k - \sum_{k=1}^{K'} W_k : \sum_{k=1}^{K'} W_k \bar{y}'_k \leq \sum_{k=1}^{K'} W_k - \lceil z^w_{LP} \rceil, \ \bar{y}' \in \mathbb{B}^{K'} \right\}.
\end{aligned}$$

This bounded knapsack problem is sometimes referred as the value independent knapsack problem or subset sum problem. It has profit coefficients equal to the weights of

the items. At the root node, we used the code presented in Martello and Toth [88] to solve it. In the other branching nodes, dynamic programming may be required, since we try to take advantage of the particular branching constraints that are enforced to strengthen the level cut. This matter is further discussed in Section 3.5.

3.4.2 The Feasibility Cuts

Another family of valid inequalities that can be used for the MLCSP are the feasibility cuts used by Vanderbeck in [122] for the case of a single stock length. These cuts are based on the fact that any set of items must obviously fit in a integer combination of rolls whose total length is at least immediately greater than the total size of the items. Here, we will derive feasibility cuts based on a single item size, since in this case the dual variables of the corresponding constraints in the master are easily reported to the pricing subproblem.

When multiple stock lengths are available, we compute the right-hand-side of the feasibility cuts using the stock rolls in decreasing order of their lengths. If the availability of the largest roll $k = 1$ is enough to cope with all the items of size w_i, we will have

$$\sum_{k=1}^{K} \sum_{p \in P_k : a_{ip}^k > 0} \lambda_p^k \geq \left\lceil \frac{b_i}{\left\lfloor \frac{W_1}{w_i} \right\rfloor} \right\rceil .$$

On the other hand, if the following holds

$$\left\lceil \frac{b_i}{\left\lfloor \frac{W_1}{w_i} \right\rfloor} \right\rceil > B_1,$$

and the availability of the roll with size W_2 is enough to cut the remaining part that does not fit in the rolls of length W_1, the following cut will be enforced

$$\sum_{k=1}^{K} \sum_{p \in P_k : a_{ip}^k > 0} \lambda_p^k \geq B_1 + \left\lceil \frac{b_i - B_1 \left\lfloor \frac{W_1}{w_i} \right\rfloor}{\left\lfloor \frac{W_2}{w_i} \right\rfloor} \right\rceil ,$$

and so forth. Following this scheme, we derive m feasibility cuts, one for each item size.

3.5 A Note on Node Fathoming

Depending on the set of branching constraints that are enforced at a node, some improvements can be introduced: the level cut described above may eventually be strengthened, or the node may be fathomed without having to solve any pricing subproblem. These situations are described next.

For a node w, let l_k be the greatest lower bound imposed on the z_k variables of formulation (3.5)-(3.10). If there is no such branching constraint for a stock length k, we will have implicitly $l_k = 0$. Let also z_{UB} be the value of the current incumbent. If the following inequality holds

$$\sum_{k \in K} l_k W_k \geq z_{UB}, \tag{3.15}$$

the node will surely never lead to an improving solution. It can then be fathomed. If this comparison is made before the node is created, computational savings result by preventing the storage and handling of the data structures for an uninteresting problem. Similarly, let u_k be an upper bound imposed through a branching constraint on variable z_k. Again, if no branching constraint is enforced at node w on z_k, we will have $u_k = B_k$. If the following holds

$$\sum_{k \in K} u_k W_k < z_{IP}^w,$$

the problem is infeasible, and the corresponding node can also be avoided.

When lower bounds l_k are effectively enforced at a node w on the z_k variables, for a subset of stock rolls say K', the level cut may possibly be strengthened if the value of its right hand side z_{IP}^w depends on a combination of rolls that does not include one, or more, of the rolls in K'. Hence, prior to the resolution of the first LP relaxation of this node w, we recompute z_{IP}^w taking into account these compulsory rolls, compare it with the incumbent, and proceed with column generation only if the node cannot be pruned.

For a node w, if there is a lower bound l_{ij} enforced on an arc (i, j) such that $i > W_2$, the second largest roll, there will be implicitly an identical constraint on the z_1 variable. If this lower bound dominates the other bounds that might have been imposed on z_1, it must be used instead, in order to evaluate the feasibility of the node quickly, or to strengthen the level cut. Generally, if $i \leq W_2$, we will not be able to anticipate to which precise roll will the items go, but we can still derive a general lower bound for the z_k variables with $W_k > i$ that can be used in (3.15) together with the length of the smallest roll greater than i.

3.6 Computational Experiments

We coded our algorithm using the C++ language, and used the CPLEX Callable Library (version 6.5) [69] to implement some of the optimization subroutines. The experiments were run on a 700 MHz PentiumIII with 128 MBytes of RAM.

We consider three different sets of instances. In the first set, the roll lengths and item sizes are random values drawn from an uniform distribution in the intervals [100,150]

and [20,100], respectively. The rolls' availability is bounded, and the demands are typically small. The second set is composed by 300 instances with a maximum of 5 different stock lengths. These instances were generated by Monaci that used them to evaluate the performance of his exact algorithm [90]. This set is further divided in three groups according to the intervals in which the item sizes vary, respectively [1,100], [20,100] and [50,100]. The roll lengths are integer values belonging to the set {60,80,100,120,150}. No restrictions are imposed on the availability of the rolls. The final set of instances is due to Belov and Scheithauer [13]. We consider the instances for the basic problem type described in this paper. The instances have at most 4 different roll lengths, with the largest reaching 10000 units and the smallest 5000. On average, these instances have 100 different items with demands varying between 1 and 100.

In the subsequent tables, we employ the following notation

. m: number of different item sizes;

. m': total number of items ($m' = \sum_i w_i b_i$);

. k: number of different roll lengths;

. $cols_{IN}$: number of columns before column generation;

. sp_{LP}: number of pricing subproblems solved before branching;

. $cols_{LP}$: number of generated columns during the resolution of the LP relaxation;

. sp_{BB}: number of pricing subproblems solved in the branch-and-bound phase;

. $cols_{BB}$: number of generated columns in the branch-and-bound phase;

. nod_{BB}: number of branching nodes covered during branch-and-bound;

. t_{PP}: time in seconds spent with preprocessing (FFD type heuristic);

. t_{LP}: solution time in seconds for the LP relaxation;

. t_{BB}: time in seconds spent with branch-and-bound;

. t_{TOT}: total computing time in seconds;

. z_{wb}: total size of the ordered items;

. z_{WB}: total roll length that is available;

. z_{LP} cost of the LP solution;

. z^*: value of the optimum integer solution.

Tables 3.1, 3.2 and 3.3 provide the computational results obtained with the 60 random instances. There are 20 instances divided in three groups, namely vs3, vs5 and vs6. Due to their random nature and the constraints on the availability of the rolls, the generated problems may be infeasible. This is what happens with the instance vs516. For these sets of instances, the roll lengths are quite diversified. In the first and second sets, the total roll length available is not much larger than the total size of the items, while in the third set, there is a greater choice of rolls in which to cut the ordered items. As a consequence, for vs6, both the differences $z_{LP} - z_{wb}$ and $z^* - z_{LP}$ are tiny, and cutting plans can be found that have almost no trim loss.

Sometimes, the optimum integer solution is found without having to resort to branch-and-bound. In fact, two cases arise: the solution to the LP relaxation of (3.1)-(3.4) is already integer (vs611, vs305 and vs315), or the integer optimum is obtained after strengthening the LP relaxation with the valid inequalities presented above (vs300, vs501 and vs509). Usually, for the instances in the latter case, the integer optimum is equal to z_{WB}, and all the capacity is used to cut the items.

The average computing times are very small, barely greater than one second for vs5. The same instances were solved using the LP relaxation of the arc flow model at each node of the branch-and-bound tree through a procedure very similar to the one described in [115], and the results were clearly worse.

Tables 3.4, 3.5 and 3.6 illustrate the average results obtained with the 300 instances of Monaci. The results are presented as in [90]. Within a time limit of 900 seconds, Monaci was able to solve only 78% of the instances, while with the algorithm described in this chapter, all the instances were solved to optimality in less than 5 seconds, on average. Even more surprising is the fact that half the instances of the last set are solved without resorting to branch-and-bound, only with the simple rounding heuristic presented above.

The last results are compiled in Table 3.7. From a set of 50 instances, 90% were solved to optimality within a time limit of 900 seconds. Only the computational results for these instances are listed. On average, they take half a minute to solve. The number of branching nodes visited is at most 242, for the 35^{th} instance. For the 5 instances that were not solved, we were able to get a proven optimality gap extremely small. On average, this gap is never greater than 0.0025%, and is obtained after 12.5 seconds spent in 10.7 branching nodes, on average.

Name	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	z_{wb}	z_{LP}	z^*	z_{WB}
vs300	17	9	12	12	88	0	0	0	0.06	0.03	0.00	0.09	1269	1276.42	1371	1371
vs301	17	9	10	15	112	141	121	100	0.06	0.05	1.04	1.15	1030	1030.00	1031	1450
vs302	16	11	12	11	79	0	0	0	0.05	0.06	0.00	0.11	1256	1271.62	1353	1353
vs303	19	11	11	14	131	1	0	1	0.05	0.06	0.00	0.11	1068	1068.00	1068	1289
vs304	15	11	12	13	94	7	3	6	0.06	0.05	0.06	0.17	1142	1144.13	1177	1318
vs305	19	10	12	16	128	0	0	0	0.06	0.05	0.00	0.11	1164	1171.00	1171	1283
vs306	17	11	12	13	104	6	9	4	0.05	0.06	0.05	0.16	1245	1248.88	1259	1408
vs307	18	11	11	12	91	1	0	1	0.06	0.05	0.00	0.11	1049	1049.00	1071	1351
vs308	18	9	12	20	123	6	3	5	0.05	0.11	0.00	0.16	1176	1179.44	1182	1386
vs309	17	9	12	13	81	0	0	0	0.05	0.06	0.00	0.11	1280	1294.25	1385	1385
vs310	19	9	12	15	105	2	0	2	0.05	0.05	0.00	0.10	1182	1183.94	1203	1350
vs311	18	10	12	14	94	49	43	40	0.05	0.06	0.28	0.39	1242	1247.55	1258	1358
vs312	20	10	12	14	101	1	0	1	0.05	0.06	0.05	0.16	1199	1199.25	1222	1367
vs313	20	11	12	17	121	169	162	119	0.05	0.06	1.10	1.21	1173	1173.40	1178	1312
vs314	17	9	11	15	109	11	7	8	0.06	0.05	0.06	0.17	1201	1208.67	1214	1342
vs315	19	8	12	16	89	0	0	0	0.05	0.06	0.00	0.11	1298	1344.00	1344	1344
vs316	19	11	11	15	130	328	226	252	0.05	0.06	2.74	2.85	1094	1094.00	1095	1354
vs317	17	11	12	7	61	0	0	0	0.05	0.06	0.00	0.11	1275	1391.00	1391	1391
vs318	18	10	12	12	81	0	0	0	0.06	0.05	0.00	0.11	1273	1307.94	1314	1314
vs319	18	10	12	11	86	46	73	35	0.06	0.05	0.28	0.39	1229	1234.00	1243	1358
avg.	17.90	10.00	11.70	13.75	100.40	38.40	32.35	28.70	0.05	0.06	0.28	0.39	1192.25	1205.82	1226.50	1354.20

Table 3.1: Computational results for the group vs3 of random instances

Name	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	z_{wb}	z_{LP}	z^*	z_{WB}
vs500	26	14	17	21	222	2	0	2	0.11	0.08	0.02	0.21	1869	1869.96	1891	2039
vs501	25	12	17	17	134	0	0	0	0.05	0.06	0.00	0.11	1833	1837.07	1924	1924
vs502	25	14	16	15	189	4	24	2	0.09	0.07	0.05	0.21	1668	1668.00	1668	1883
vs503	27	13	16	16	180	56	155	36	0.11	0.10	0.71	0.92	1626	1626.00	1628	1940
vs504	23	15	17	14	172	378	298	295	0.04	0.09	4.84	4.97	1798	1798.00	1798	2042
vs505	26	13	17	16	150	5	1	4	0.05	0.06	0.06	0.17	1734	1734.85	1753	1900
vs506	24	13	17	20	186	4	0	4	0.11	0.11	0.06	0.28	1830	1834.30	1889	2037
vs507	25	15	17	17	194	4	4	2	0.09	0.12	0.05	0.26	1792	1792.00	1831	1975
vs508	29	15	16	16	206	43	93	36	0.16	0.09	0.55	0.80	1700	1700.00	1704	2024
vs509	24	15	17	16	180	0	0	0	0.11	0.07	0.00	0.18	1885	1891.13	1958	1958
vs510	28	12	17	17	142	0	0	0	0.05	0.11	0.00	0.16	1872	1880.67	1904	1904
vs511	26	16	17	15	178	72	56	48	0.08	0.05	0.66	0.79	1832	1832.65	1836	1963
vs512	24	14	17	15	155	0	0	0	0.05	0.10	0.00	0.15	1865	1872.38	1938	1938
vs513	26	13	16	16	175	8	12	7	0.12	0.13	0.11	0.36	1759	1759.00	1760	1994
vs514	27	14	17	17	197	0	0	0	0.09	0.14	0.00	0.23	1781	1781.00	1804	1952
vs515	25	12	16	15	154	160	179	120	0.11	0.11	1.63	1.85	1716	1716.00	1718	1966
vs517	26	13	17	16	140	0	0	0	0.10	0.12	0.00	0.22	1882	1884.35	1945	1945
vs518	26	15	16	13	180	713	550	570	0.05	0.07	10.63	10.75	1644	1644.00	1644	1886
vs519	27	15	17	13	155	22	53	13	0.07	0.11	0.22	0.40	1825	1825.00	1825	1941
avg.	25.74	13.84	16.68	16.05	173.11	77.42	75.00	59.95	0.09	0.09	1.03	1.21	1784.79	1786.65	1811.47	1958.47

Table 3.2: Computational results for the group vs5 of random instances

Name	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	z_{wb}	z_{LP}	z^*	z_{WB}
vs600	18	14	13	14	125	76	62	56	0.11	0.06	0.43	0.60	1279	1280.80	1282	2039
vs601	19	12	12	15	136	44	66	28	0.05	0.05	0.28	0.38	1115	1115.00	1115	1934
vs602	17	12	11	11	110	198	161	152	0.06	0.05	1.43	1.54	1031	1031.00	1032	2045
vs603	18	15	12	13	162	17	45	10	0.05	0.11	0.05	0.21	1115	1115.00	1115	1935
vs604	19	12	13	13	133	76	96	47	0.05	0.05	0.49	0.59	1138	1138.00	1138	1901
vs605	17	14	13	13	123	39	41	26	0.06	0.00	0.22	0.28	1274	1276.00	1277	2009
vs606	18	16	11	11	145	5	12	4	0.05	0.06	0.05	0.16	1084	1084.00	1084	1983
vs607	19	13	12	13	150	3	12	1	0.05	0.11	0.00	0.16	1128	1130.00	1130	1991
vs608	17	15	12	15	132	3	3	1	0.05	0.06	0.05	0.16	1223	1224.56	1225	1996
vs609	17	14	12	13	128	88	140	60	0.05	0.06	0.48	0.59	1169	1172.00	1172	2019
vs610	19	14	12	11	128	3	5	2	0.05	0.05	0.00	0.10	1177	1177.00	1177	2045
vs611	19	15	11	9	118	0	0	0	0.05	0.06	0.00	0.11	1082	1082.00	1082	1958
vs612	17	12	12	16	134	77	59	54	0.06	0.05	0.44	0.55	1216	1216.90	1220	1971
vs613	19	12	13	12	99	14	19	9	0.11	0.05	0.06	0.22	1262	1263.17	1264	2016
vs614	19	15	12	15	167	34	70	21	0.05	0.05	0.22	0.32	1114	1114.00	1114	1938
vs615	15	14	11	9	100	4	10	3	0.05	0.06	0.05	0.16	1071	1071.00	1071	2083
vs616	17	13	13	12	109	29	57	16	0.06	0.05	0.17	0.28	1206	1208.94	1212	2016
vs617	17	14	11	10	110	2	8	1	0.06	0.05	0.00	0.11	1137	1137.00	1137	2041
vs618	20	15	13	10	124	26	22	16	0.11	0.05	0.17	0.33	1258	1260.00	1260	1955
vs619	18	13	13	11	103	3	4	2	0.05	0.06	0.05	0.16	1209	1212.00	1213	1908
avg.	17.95	13.70	12.10	12.30	126.80	37.05	44.60	25.45	0.06	0.06	0.23	0.35	1164.40	1165.42	1166.00	1989.15

Table 3.3: Computational results for the group vs6 of random instances

m	m'	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}
22.00	25	3	14.60	23.50	52.20	22.30	18.70	11.60	0.12	0.08	0.13	0.33
38.30	50	3	26.60	37.90	98.30	12.40	13.90	6.10	0.21	0.19	0.11	0.51
62.30	100	3	45.30	45.30	129.80	16.90	22.90	9.00	0.52	0.35	0.33	1.20
86.80	200	3	69.60	63.90	185.60	38.30	43.60	23.50	0.97	0.70	1.32	3.00
98.80	500	3	89.30	70.10	203.80	29.40	14.30	24.50	1.70	1.07	2.00	4.76
22.40	25	5	18.10	21.60	68.90	7.80	6.30	4.50	0.10	0.08	0.05	0.23
39.10	50	5	31.90	35.90	125.30	3.60	3.00	1.80	0.32	0.19	0.04	0.56
61.80	100	5	53.20	38.50	165.40	16.30	22.20	7.80	0.76	0.31	0.28	1.35
85.60	200	5	76.10	47.40	216.80	16.70	23.40	11.30	1.48	0.57	0.53	2.58
98.50	500	5	93.30	52.00	245.70	10.00	4.40	9.00	2.19	0.86	0.59	3.64

Table 3.4: Computational results for the first group of instances from [90]

m	m'	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}
21.60	25	3	17.30	21.30	46.30	9.50	7.00	5.00	0.12	0.07	0.04	0.23
36.90	50	3	29.10	46.90	99.50	13.00	10.80	6.30	0.25	0.20	0.08	0.52
58.70	100	3	49.80	71.40	168.00	15.40	14.40	6.00	0.61	0.44	0.17	1.21
73.10	200	3	65.60	73.10	198.10	15.50	11.90	9.00	1.03	0.61	0.33	1.97
80.00	500	3	77.00	61.40	173.30	23.40	16.20	17.20	1.47	0.84	0.68	2.99
22.20	25	5	21.80	20.60	55.80	4.10	3.90	1.70	0.14	0.06	0.02	0.21
37.40	50	5	36.30	44.20	119.30	14.20	8.60	7.30	0.40	0.19	0.12	0.71
57.50	100	5	56.00	66.00	189.20	28.20	15.80	16.80	0.86	0.42	0.40	1.67
73.70	200	5	72.90	62.60	232.60	23.20	23.30	11.40	1.44	0.56	0.50	2.51
79.70	500	5	79.30	64.90	245.80	20.30	13.90	15.30	1.79	0.72	0.62	3.13

Table 3.5: Computational results for the second group of instances from [90]

m	m'	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}
19.40	25	3	20.40	23.60	39.50	0.70	0.40	0.30	0.15	0.05	0.01	0.20
30.70	50	3	31.70	41.70	76.50	8.80	5.40	4.40	0.29	0.11	0.04	0.44
44.80	100	3	45.80	74.00	138.00	55.30	8.70	46.80	0.60	0.25	0.48	1.34
49.30	200	3	50.30	82.20	163.80	90.10	7.70	83.30	0.71	0.34	0.95	2.00
50.00	500	3	51.00	74.44	155.33	61.56	6.00	56.89	0.99	0.32	0.59	1.90
18.50	25	5	19.50	23.70	38.60	0.00	0.00	0.00	0.11	0.04	0.00	0.15
31.60	50	5	32.60	43.80	79.60	0.00	0.00	0.00	0.34	0.12	0.00	0.46
43.30	100	5	44.30	58.00	122.20	0.00	0.00	0.00	0.54	0.23	0.00	0.77
49.50	200	5	50.50	68.30	160.80	0.00	0.00	0.00	0.73	0.31	0.00	1.04
50.00	500	5	51.00	71.20	164.80	0.00	0.00	0.00	0.77	0.32	0.00	1.09

Table 3.6: Computational results for the third group of instances from [90]

$Inst$	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}
2	100	3	102	107	288	0	0	0	0.98	1.13	0.00	2.11
4	99	4	117	134	421	8	0	8	1.13	1.80	5.77	8.69
5	100	4	104	131	451	0	0	0	0.95	1.44	0.00	2.39
6	99	4	104	146	497	64	35	44	0.95	1.61	45.33	47.89
7	99	4	105	136	422	47	8	43	1.14	1.49	35.17	37.80
8	99	3	105	99	263	20	9	15	1.20	0.38	14.47	16.05
9	100	4	112	121	411	38	13	31	0.94	1.28	23.97	26.19
10	99	4	110	112	409	7	2	5	1.02	1.39	4.61	7.01
11	98	4	100	114	380	12	0	12	1.13	0.81	6.13	8.06
12	98	4	123	137	434	49	15	41	1.41	1.74	39.11	42.25
14	99	4	110	118	378	5	0	5	1.19	1.31	2.62	5.12
15	100	4	111	124	437	15	9	10	0.89	5.08	10.69	16.65
16	100	3	110	187	472	23	0	23	1.27	3.47	19.03	23.76
17	100	2	118	152	302	25	7	20	1.11	2.28	19.77	23.17
18	100	3	101	121	343	12	0	12	0.97	2.06	9.18	12.22
19	99	4	107	126	378	16	6	13	1.16	0.86	11.20	13.22
20	99	4	113	134	458	82	43	64	1.74	2.03	66.28	70.06
21	98	4	104	107	371	11	7	8	0.80	1.61	8.13	10.53
22	99	3	104	123	333	14	0	14	1.05	0.91	7.85	9.80
23	98	4	109	127	432	30	1	29	0.95	4.06	22.75	27.77
24	99	4	109	61	225	0	0	0	1.09	0.33	0.00	1.42
25	100	4	112	161	487	29	9	25	1.25	1.66	22.08	24.98
27	99	4	111	98	335	6	0	6	0.92	1.14	2.73	4.80
28	98	3	104	122	306	27	2	25	0.73	1.17	29.70	31.61
29	99	4	121	142	508	8	0	8	0.94	2.95	6.84	10.73
30	99	4	117	127	437	63	27	52	1.23	2.40	51.50	55.14
31	99	4	106	96	328	14	3	11	0.72	0.94	8.03	9.69
32	99	4	106	113	406	53	21	40	1.16	1.01	47.38	49.55
33	98	4	116	104	360	42	29	33	1.22	0.56	19.30	21.08
34	100	4	111	145	470	53	12	47	1.41	2.42	55.44	59.26
35	100	4	111	187	612	298	96	242	1.06	6.45	313.83	321.34
36	100	4	117	105	395	30	13	24	1.13	2.87	23.79	27.79
37	100	4	114	115	396	76	30	61	1.11	1.72	59.98	62.81
38	100	4	120	95	342	4	0	4	1.14	1.11	2.83	5.08
39	98	4	107	118	417	2	0	2	0.91	1.11	0.91	2.92
40	99	4	107	126	420	8	0	8	1.27	1.42	5.13	7.83
41	100	4	111	164	513	159	60	125	1.17	1.48	117.77	120.42
42	98	3	99	230	402	0	0	0	1.06	1.12	0.00	2.19
43	99	4	108	134	352	41	12	32	1.27	0.86	34.69	36.81
44	99	4	110	92	321	11	1	10	1.23	0.66	7.48	9.37
45	100	4	110	76	232	5	10	5	1.13	0.25	3.85	5.22
46	99	2	99	45	63	16	2	15	0.88	0.06	11.92	12.86
47	98	4	101	104	344	5	3	4	0.95	0.94	3.61	5.50
48	98	4	104	168	525	20	0	20	0.97	2.55	14.44	17.96
49	99	4	105	102	363	56	35	41	1.08	1.31	48.42	50.81
avg.	99.11	3.76	109.00	124.13	387.53	33.42	11.56	27.49	1.09	1.67	27.64	30.40

Table 3.7: Computational results for a set of instances used in [13]

3.7 The Combined Assortment and Trim Loss Minimization Problem

3.7.1 Introduction

A problem that is directly related to the multiple length cutting stock problem is the one of selecting from a set of possible stock lengths the assortment that will be used in the cutting plan. Hinxman [64] distinguished between the assortment and the trim loss minimization problem, but both can be addressed in the same problem. In this chapter, we will extend the approach described above for the MLCSP to the problem that we will here designate by combined assortment and trim loss minimization problem. After describing the modifications done to the algorithm, we report on the results obtained for various random instances.

In [64], Hinxman surveyed the main solution approaches devised for the assortment problem. In their majority, these approaches are based on dynamic programming. The number of recent publications regarding this problem is not high. Another dynamic programming algorithm was proposed by Baker [7] for a special case. More recently, Holthaus [68] investigated the impact of having assortments with more than one stock length, analyzing many problem instances.

3.7.2 An Integer Programming Formulation

Extending the column generation model (3.1)-(3.4) to the combined assortment and trim loss minimization problem is straightforward. A set of binary variables, one for each stock length, has to be added to the model, and a new constraint is enforced, in order to limit to K' the number of different lengths that can be effectively used in the cutting plan. The resulting model is as follows

$$\min \quad \sum_{k=1}^{K} \sum_{p \in P^k} W_k \lambda_p^k \tag{3.16}$$

$$\text{subject to} \quad \sum_{k=1}^{K} \sum_{p \in P^k} a_{ip}^k \lambda_p^k \geq b_i, \ i = 1, ..., m, \tag{3.17}$$

$$\sum_{p \in P_k} \lambda_p^k \leq B_k \mu_k, \ k = 1, ..., K, \tag{3.18}$$

$$\sum_{k=1}^{K} \mu^k \leq K', \tag{3.19}$$

$$\lambda_p^k \geq 0, \text{ and integer}, \ k = 1, \ldots, K, \ p \in P^k, \tag{3.20}$$

$$\mu_k \in \{0, 1\}, \ k = 1, \ldots, K. \tag{3.21}$$

For the problems without constraints on the number of available rolls (3.18), the LP bound provided by this model is typically weak. These constraints are a first way of getting a stronger model, but to further strengthen the model we essentially rely on an adaptation of the level cut described above.

Model (3.16)-(3.21) is tackled with column generation, in a way very similar to the one used for the MLCSP. The LP master has K additional columns related to the binary variables, but to price out an attractive pattern, we still have to solve a knapsack problem. The values for the dual variables associated to the constraint (3.19) have no incidence in the dual price of the columns that represent cutting patterns, since these columns have a null coefficient in this constraint.

Before describing the modifications for the computation of the level cut, we first present the details of the branching scheme employed.

3.7.3 Branch-and-Price

At each node of the branch-and-bound tree, we solve the LP relaxation of (3.16)-(3.21). We start with a model that has one artificial column, K columns for the binary variables as referred above, and a set of initial columns, generated using a FFD heuristic that only takes into account the K' largest stock lengths.

An optimal LP solution may be fractional because of its pattern frequencies, or because one or more of the binary variables is neither 0 nor 1. More than K' of the μ_k variables may even be non-negative. We will first branch on these variables. Branching on the μ_k variables does never affect the pricing subproblem, so we can freely define any branching constraint on them. If, say, μ_k is fractional, two nodes are created with one of the following branching constraints

$$\mu_k = 0, \tag{3.22}$$

and

$$\mu_k = 1. \tag{3.23}$$

With (3.22), the stock length is excluded from the set of possible rolls, and hence, it can be simply removed from the instance. With (3.23), what we force is the accounting of k as an used stock length. However, in practice, this stock length may not be used. What we guarantee is that only $K' - 1$ stock lengths other than k can effectively be used. If all the variables μ_k are integer, we proceed with the branching scheme described in Section 3.3.2. The LP solution is converted into a set of flows in the arc flow model, and the z_k variables are checked for integrality, followed by variables x_{ij}.

The node selection is done in a depth first manner. At most two nodes are created when branching, and the one with (3.23) or a "greater than or equal to" branching constraint is selected first.

At the root node, there are still K knapsack subproblems to solve per iteration of the column generation procedure. For the other nodes, even the ones where constraints of type (3.22) are enforced, attractive patterns can still be found in a single run of a dynamic programming algorithm.

3.7.4 Extending the Level Cut

In the combined assortment and trim loss minimization problem, the items have to be assigned to a set of rolls from a bounded subset of the available stock lengths. Hence, the cost of a solution must correspond to a non-negative combination of at most K' different stock lengths. This fact helps in getting stronger level cuts.

At each node of the branch-and-bound tree, the right hand side of the level cut is computed with a dynamic programming algorithm. In the state space, the number of stock lengths used to reach a specific level must now be registered. Therefore, we have states defined as the pairs $(n, level)$, where n is the number of stock lengths used, and $level$ identifies a reachable length obtained by combining no more than K' different stock lengths.

In the computational implementation of our dynamic programming algorithm, the lengths for which there is a branching constraint of type (3.23) are treated first. Including the lengths in this situation to the combination is not compulsory, but the transition between two states in the stages associated to these lengths is always done from a state (n, l_1) to another state $(n+1, l_2)$, with l_1 not necessarily different from l_2. The lengths for which a branching constraint of type (3.22) has been enforced are removed from the instance, and obviously they are not considered in the computation of the level cut.

3.7.5 Computational Experiments

A set of computational experiments were conducted on 160 random instances. These tests were performed on a 3GHz Pentium IV computer with 512MBytes of RAM. To generate the test problems, we used the CUTGEN1 generator described in [50], with a seed equal to 1994. Sixteen groups of ten instances were used. Their main characteristics are summarized in Table 3.8. The instances have at most 50 different item sizes and between 5 and 20 stock lengths in the interval $[100, 300]$. The average demand per item type is always 10 units. Hence, for the instances with $m = 50$, for example, there will be a total of 500 items to cut from the rolls. In the subsequent tables, m represents the parameter of the CUTGEN1 generator that is related to the number of item sizes, while $\overline{m}$ is the real average number of different item sizes in the instances.

Set	m	K	v_1	v_2	$\bar{b}$
1	20	5	0.1	0.8	10
2		10			
3		15			
4		20			
5	30	5	0.1	0.8	10
6		10			
7		15			
8		20			
9	40	5	0.1	0.8	10
10		10			
11		15			
12		20			
13	50	5	0.1	0.8	10
14		10			
15		15			
16		20			

Table 3.8: Characteristics of the random instances

For each problem set, our algorithm was run four times. In the first run, all the K stock lengths can be used, and the problems reduce to the MLCSP. In the remaining three runs, we restrict the set of stock lengths to 75%, 50% and 25% of K, respectively. In the subsequent tables, the column designated by K' identifies the respective percentage of permitted stock lengths.

Tables 3.9, 3.10, 3.11 and 3.12 report on the average computational results obtained with each problem set. Note that these averages do not take into account the instances that were not solved within a time limit of 900 seconds. In 640 problems to solve, our algorithm found an optimal integer solution within the time limit in 97% of the cases. Column *opt* indicates the number of instances solved to optimality within each set. On average, the computing times are rather small, and some of the high values that appear are essentially due to a very small number of instances in the sets for which the algorithm performed poorly.

3.8 Conclusion

The cutting stock problem with multiple lengths is harder than the standard problem. With more stock lengths available, there are also more feasible cutting patterns. Furthermore, the continuous bound given by the column generation model is not as tight as the one for the cutting stock problem, for which the integer round-up property applies. An interesting work would be to study possible extensions of this round-up property to the problem with multiple lengths.

Set	$\overline{m}$	K	K'	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	opt
1	17.20	5.00	100	23.80	15.00	60.70	13.20	6.50	11.50	0.02	0.01	0.10	0.13	10
	17.20	5.00	75	23.80	15.00	60.70	30.40	33.60	21.30	0.02	0.01	0.15	0.17	10
	17.20	5.00	50	23.80	15.00	60.70	36.60	31.50	26.40	0.02	0.01	0.14	0.17	10
	17.20	5.00	25	23.80	15.00	60.70	83.80	46.20	62.80	0.02	0.01	0.17	0.20	10
2	17.20	10.00	100	28.80	10.30	87.50	33.80	43.00	28.10	0.02	0.01	0.71	0.75	10
	17.20	10.00	75	28.80	10.30	87.50	41.60	93.20	29.50	0.02	0.01	0.62	0.65	10
	17.20	10.00	50	28.80	10.30	87.50	55.50	125.90	31.50	0.02	0.01	0.39	0.42	10
	17.20	10.00	25	28.80	10.30	87.50	72.10	150.40	35.70	0.02	0.01	0.32	0.35	10
3	17.20	15.00	100	33.80	10.00	130.70	54.50	112.50	44.00	0.02	0.02	2.15	2.19	10
	17.11	15.00	75	33.67	10.00	130.56	57.67	137.00	43.33	0.02	0.02	1.59	1.63	9
	17.20	15.00	50	33.80	10.00	130.70	99.80	239.80	57.80	0.03	0.02	1.39	1.44	10
	17.20	15.00	25	33.80	10.00	130.70	139.40	263.40	83.20	0.02	0.02	1.03	1.08	10
4	17.20	20.00	100	38.80	9.20	161.10	86.40	190.50	71.50	0.02	0.04	5.53	5.59	10
	16.88	20.00	75	38.88	9.13	159.00	97.00	242.88	73.38	0.02	0.02	4.21	4.25	8
	17.20	20.00	50	38.80	9.20	161.10	362.50	268.40	262.70	0.03	0.03	7.70	7.75	10
	17.22	20.00	25	38.78	9.56	168.00	162.89	499.22	80.00	0.02	0.03	1.81	1.86	9

Table 3.9: Computational results for the random instances with $m = 20$

Set	$\overline{m}$	K	K'	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	opt
5	25.00	5.00	100	33.50	15.80	70.90	26.70	30.00	19.60	0.05	0.02	0.30	0.36	10
	25.00	5.00	75	33.50	15.80	70.90	43.00	68.70	24.40	0.05	0.01	0.29	0.36	10
	25.00	5.00	50	33.50	15.80	70.90	37.50	58.10	19.70	0.05	0.02	0.19	0.26	10
	25.00	5.00	25	33.50	15.80	70.90	118.00	146.00	59.50	0.05	0.01	1.34	0.60	10
6	25.00	10.00	100	38.50	12.30	112.30	51.80	77.70	42.40	0.05	0.03	1.78	1.85	10
	25.00	10.00	75	38.50	12.30	112.30	68.50	160.70	46.90	0.04	0.03	1.56	1.63	10
	25.00	10.00	50	38.50	12.30	112.30	152.30	296.20	83.40	0.04	0.03	1.71	1.78	10
	25.00	10.00	25	38.50	12.30	112.30	130.80	288.30	58.80	0.05	0.03	0.98	1.06	10
7	25.00	15.00	100	43.50	11.30	153.90	255.90	201.60	234.70	0.05	0.05	15.76	15.86	10
	25.00	15.00	75	43.50	11.30	153.90	99.90	316.70	72.50	0.05	0.04	4.58	4.68	10
	25.00	15.00	50	43.44	11.00	149.56	1030.67	547.56	710.22	0.05	0.05	25.46	25.56	9
	25.00	15.00	25	43.50	11.30	153.90	291.40	862.10	132.40	0.06	0.04	4.05	4.15	10
8	25.00	20.00	100	48.50	10.60	191.40	111.00	332.80	87.10	0.05	0.07	10.96	11.07	10
	25.00	20.00	75	48.50	10.60	191.40	126.30	468.80	93.00	0.05	0.07	8.39	8.50	10
	24.63	20.00	50	48.00	10.13	182.00	223.63	619.50	118.88	0.05	0.07	6.97	7.09	8
	25.00	20.00	25	48.50	10.60	191.40	765.00	1552.10	418.60	0.05	0.07	17.54	17.66	10

Table 3.10: Computational results for the random instances with $m = 30$

Set	$\overline{m}$	K	K'	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	*opt*
9	30.00	5.00	100	37.40	18.70	86.20	47.00	52.40	35.40	0.06	0.02	0.88	0.96	10
	30.00	5.00	75	37.40	18.70	86.20	41.70	64.80	27.30	0.06	0.02	0.55	0.63	10
	30.00	5.00	50	37.40	18.70	86.20	182.00	132.60	139.70	0.07	0.02	1.70	1.78	10
	30.00	5.00	25	37.40	18.70	86.20	115.00	169.70	48.50	0.06	0.02	0.48	0.56	10
10	30.00	10.00	100	42.40	13.70	126.70	123.70	186.30	96.70	0.06	0.04	6.23	6.34	10
	30.00	10.00	75	42.40	13.70	126.70	130.40	337.80	81.70	0.06	0.05	4.36	4.47	10
	30.00	10.00	50	42.40	13.70	126.70	145.80	367.40	67.80	0.06	0.04	2.33	2.43	10
	30.00	10.00	25	42.40	13.70	126.70	228.60	490.20	102.80	0.06	0.03	2.37	2.47	10
11	30.11	15.00	100	47.33	13.11	181.33	1015.00	389.22	972.22	0.06	0.08	98.82	98.96	9
	30.00	15.00	75	47.40	13.00	179.70	252.50	598.70	177.10	0.06	0.06	16.40	16.52	10
	30.00	15.00	50	47.40	13.00	179.70	311.50	751.90	151.70	0.06	0.08	9.62	9.76	10
	30.00	15.00	25	47.40	13.00	179.70	2063.10	1568.10	1624.00	0.06	0.09	53.15	53.29	10
12	30.11	20.00	100	52.33	12.78	234.89	188.89	603.44	137.67	0.06	0.12	26.42	26.60	9
	29.88	20.00	75	52.25	13.13	241.75	209.25	756.38	141.13	0.06	0.12	19.69	19.88	8
	29.71	20.00	50	52.14	12.00	219.14	310.71	1003.43	148.57	0.06	0.11	14.46	14.63	7
	29.78	20.00	25	52.44	12.11	221.56	556.89	2724.89	196.89	0.07	0.11	17.41	17.58	9

Table 3.11: Computational results for the random instances with $m = 40$

Set	$\overline{m}$	K	K'	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	*opt*
13	36.10	5.00	100	43.60	20.30	95.40	65.80	60.60	50.30	0.07	0.03	1.68	1.79	10
	36.10	5.00	75	43.60	20.30	95.40	83.50	131.60	49.80	0.08	0.03	1.36	1.46	10
	36.10	5.00	50	43.60	20.30	95.40	1129.70	189.20	1039.20	0.08	0.03	20.27	20.37	10
	36.10	5.00	25	43.60	20.30	95.40	112.70	191.90	41.40	0.08	0.03	0.57	0.68	10
14	36.10	10.00	100	48.60	15.50	144.30	148.40	262.80	112.90	0.08	0.06	10.71	10.84	10
	36.10	10.00	75	48.60	15.50	144.30	134.50	346.80	82.90	0.08	0.06	6.45	6.59	10
	36.00	10.00	50	48.56	15.44	143.67	257.89	587.56	111.78	0.08	0.07	5.31	5.45	9
	36.10	10.00	25	48.60	15.50	144.30	393.50	873.50	159.40	0.08	0.06	5.80	5.94	10
15	35.78	15.00	100	53.44	14.56	199.22	203.67	412.89	145.78	0.08	0.24	27.22	27.55	9
	36.10	15.00	75	53.60	14.70	200.20	278.80	688.60	170.30	0.08	0.26	26.45	26.79	10
	35.89	15.00	50	53.00	14.56	197.56	1011.56	1017.33	737.78	0.08	0.24	60.25	60.56	9
	36.10	15.00	25	53.60	14.70	200.20	1522.90	1937.90	819.10	0.08	0.18	55.86	56.12	10
16	36.10	20.00	100	58.60	14.50	263.00	273.40	624.80	192.20	0.09	0.46	57.43	57.99	10
	35.78	20.00	75	58.33	13.78	247.78	400.89	1195.78	221.89	0.10	0.45	49.30	49.84	9
	36.10	20.00	50	58.60	14.50	263.00	768.40	1746.50	368.00	0.10	0.38	50.07	50.54	10
	36.00	20.00	25	58.22	14.67	265.56	994.33	4125.00	312.00	0.09	0.30	40.76	41.15	9

Table 3.12: Computational results for the random instances with $m = 50$

The arc flow model has already been used in a branch-and-price framework to solve the standard cutting stock problem. However, in the algorithm described in [115], the formulation for the master has a considerable number of constraints. To deal with the size of the model, constraints are considered only if they are really necessary. A similar scheme was tested with instances of the multiple length cutting stock problem, but it was clearly outperformed by the algorithm explored in this chapter. In fact, using the Gilmore-Gomory formulation for the master, and deriving a branching scheme from an arc flow model results in a branch-and-price algorithm for the cutting stock problem with multiple stock lengths that also proved to be more efficient than other exact algorithms presented so far.

Experiments have also shown that the combined assortment and trim loss minimization problem can be solved with very satisfactorily results, with an extension of our previous branch-and-price algorithm. The main element contributing to these results is the adaptation of the level cut to a problem in which there is a restriction on the total number of different stock lengths that can be used.

Chapter 4

Accelerating Column Generation for Cutting Stock Problems

Column generation algorithms suffer from the well known tailing off effect, or long tail convergence. Typically, in the initial iterations, the value of the optimum is quickly improved, and then the process considerably slows down as it goes to its last iterations. A reason that is commonly pointed out to justify this phenomenon is the instability of the dual variables. Hence, much of the methods that have been proposed to accelerate column generation algorithms tries to stabilize them, in order to avoid their erratic behavior. A method that achieved this stabilization with an appreciable success consists in adding extra primal columns to the LP master, corresponding to feasible dual inequalities [117]. Until now, this method has been applied only to LP relaxations, without any interface with branch-and-bound frameworks. In this chapter, we extend the use of a family of dual inequalities for the standard cutting stock problem to a whole branch-and-bound tree, and we present new dual inequalities for the multiple length cutting stock problem. Computational results are given attesting the efficacy of the procedures.
We also propose an alternative way of enforcing inequalities in the dual by aggregating parts of the primal model. Row aggregation, and a double aggregation of rows and columns are explored. Extensive computational experiments are reported on randomly generated instances, and instances from the literature.

Keywords: Column Generation, Branch-and-Bound, Cutting Stock Problem, Multiple Length Cutting Stock Problem, Convergence, Dual-Optimal Inequalities, Model Aggregation

4.1 Introduction

Many of the algorithms for single and multiple length cutting stock problems presented in the literature rely on column generation, processes that are known to converge slowly due to primal degeneracy and the excessive oscillations of the dual variables. In the last years, many efforts have been devoted to the topic of stabilized column generation, resulting in different methods that we briefly recall in the sequel. In this chapter, we explore two different stabilization techniques: one based on enforcing valid inequalities in the dual model, and another based on model aggregation.

Usually, stabilization may be achieved by restricting once the admissible dual solution space or, alternatively, by guiding the progress of the dual variables. The boxstep method of Marsten *et al.* [87] follows this latter strategy by drawing fixed-size boxes around the solutions of the dual restricted master problem. The trust region method [74] uses a similar concept but relies on box constraints whose sizes may be dynamically updated. From the primal standpoint, these methods solve successively a restricted master problem in which slack and surplus variables are penalized.

In [35], du Merle *et al.* extend this approach by imposing additional constraints to these variables. They suggest strategies to set the box sizes and report promising results on air transportation and location problems. Other methods, such as bundle [65] and analytic center cutting plane methods [34], have been used to prevent the excessive variations of the dual variables.

Recent publications have shown how to reduce the instability of column generation when applied to cutting stock or bin-packing problems. Valério de Carvalho [117] proposes adding a polynomial number of columns (*i.e.*, dual inequalities) prior to the solution of the first restricted master problem. He derives a family of dual inequalities and proves that they are weak dual-optimal inequalities in the sense introduced by Ben Amor *et al.* [15], that is, they do not exclude any dual-optimal solution. Since primal feasibility may be lost when the corresponding primal columns are considered in the master, Ben Amor *et al.* [15] suggest perturbing the right hand side of the dual inequalities by small amounts, forcing the respective columns to have null values in any optimal solution. Furthermore, they show that the aggregation into single constraints of items with the same size leads to a substantial acceleration of the overall solution process.

In their models for packing and cutting problems, researchers have since long considered the principle of aggregation. This was already a way of implicitly restricting the dual space by enforcing equality between some dual variables. In fact, we know that there always exists an optimal solution in which items of identical sizes have corresponding dual variables with the same value. In practice, this equality extends frequently to other items with nearly the same size. This phenomenon was early pointed out

by Gilmore and Gomory [53], who explore it only to reduce the size of the knapsack subproblems.

Accelerating the resolution of the LP relaxation is an important matter, since this is a way of getting good lower bounds in less time. However, being able to extend the stabilization techniques to all the nodes of a branching tree is far more interesting. Indeed, in a branch-and-price framework the tailing off effect does not occur only at the root node. In this chapter, we study how this extension can be done. We show that adding to the LP master the columns for some dual inequalities may relax the primal problem, and we state the conditions that these dual inequalities must satisfy in order to remain valid at a specific branching node. New valid dual-optimal inequalities that take into account the dual variables of the branching constraints are also described. Additionally, dual inequalities are given for the multiple length cutting stock problem, and their relative strength is discussed.

We proceed by exploring the principle of model aggregation as an alternative way of controlling the progress of the dual variables. Two algorithms based on the iterative resolution of aggregated models are proposed. The first is a simple two-steps procedure that starts by solving a row aggregated LP resulting from the juxtaposition of the original items. The second is an iterative algorithm that solves a sequence of smaller size approximations obtained through a double aggregation of variables and constraints. This aggregation scheme amounts to imposing equality constraints between some dual variables. Note that these aggregation strategies rely essentially on items, which allows us to use them for the standard cutting stock problem.

This chapter is organized as follows. The mathematical formulation for the dual of the multiple length cutting stock problem is presented next. In Section 4.3, we review the dual inequalities introduced in [117], which still apply to the case of multiple lengths, and we describe new valid dual inequalities. In Section 4.4, we study the extension of some dual-optimal inequalities to all the nodes of a branching tree. The specific branch-and-bound framework introduced in the previous chapter is assumed. New valid dual inequalities are also described, and followed by intermediate computational results given at this point to illustrate the impact of using dual cuts in a whole branching tree. Section 4.6 is devoted to the topic of aggregation. Extensive computational results are finally reported in Section 4.7.

4.2 The Dual Formulation

Most of the results presented in this chapter are based on the dual model for the LP relaxation of (3.1)-(3.4). In the sequel, we hence review the main elements of this dual problem. Let u_i, $i = 1, \ldots, m$, and v_k, $k = 1, \ldots, K$, be the dual variables associated respectively to the demand constraints (3.2) and to the rolls' availability constraints

(3.3) of the column generation model for the MLCSP presented in the previous chapter. For ease of presentation, we will frequently refer to the u_i and v_k variables respectively as the items' and rolls' dual variables.

Based on the complementary slackness conditions, we can give an interesting interpretation to the items' dual variables. If the rolls' availability was unbounded, the u_i variables would represent the exact ideal sizes the items should have in order to fulfill the rolls selected in the primal solution. By "ideal", we mean that with these "dual sizes" the optimal cutting plan will have no trim loss. In the bounded case, when the availability constraints are effective, the v_k variables relax the dual knapsack constraints to some point. Let P denote the LP primal formulation related to (3.1)-(3.4). The dual formulation D of P follows.

$$(D) \qquad \max \sum_{i=1}^{m} u_i b_i + \sum_{k=1}^{K} v_k B_k \tag{4.1}$$

$$\text{subject to} \sum_{i=1}^{m} a_{ip}^k u_i + v_k \leq W_k, \ k = 1, \ldots, K, \ p \in P^k, \tag{4.2}$$

$$u_i \geq 0, \ i = 1, \ldots, m, \tag{4.3}$$

$$v_k \leq 0, \ k = 1, \ldots, K. \tag{4.4}$$

The u_i variables are positive in the dual formulation since they are related to demand constraints that are expressed in the primal as "greater than or equal to" constraints. On the other hand, with "less than or equal to" rolls' availability constraints, the v_k variables can only be non-positive. The column generation model (3.1)-(3.4) has an exponential number of columns. As a consequence, the dual model (4.1)-(4.4) has an exponential number of constraints. In the dual, the generic column generation algorithm can be viewed as a cutting plane procedure [77].

4.3 Dual-optimal Inequalities

In [117], Valério de Carvalho introduced a certain concept of dual cuts, columns in the primal that do not affect the optimal value as long as a solution to the original problem can be recovered at no cost with those columns at the zero level. Later on, Ben Amor *et al.* [15] used the term dual-optimal inequalities and distinguished between weak and deep inequalities. In the former, no optimal solution of the original formulation is excluded, whereas the deep inequalities may cause the rejection of some alternative dual-optimal solutions.

The cuts presented in [117] apply to the multiple length cutting stock problem. We recall them briefly in Section 4.3.1. In Section 4.3.2, we introduce new types of weak

and deep dual-optimal inequalities specially devised for the multiple length cutting stock problem.

4.3.1 Inequalities on Items' Dual Variables

If we observe the values of an optimal solution to (4.1)-(4.4), the ordering of the items' dual variables will be quite evident. After all, the items' dual variables are subject to constraints whose coefficients are directly related to the set of item sizes. This leads to an ordering that is completely dependent on the size of the items. Let S be the set of items of size w_s, such that $\sum_{s \in S} w_s \leq w_i$, for some $i = 1, ..., m$. The relation can be expressed as follows

$$-u_i + \sum_{s \in S} u_s \leq 0, \qquad i = 1, ..., m, \ \forall S. \tag{4.5}$$

Depending on the cardinality of S, different types of dual inequalities may be defined. The result is an exponential number of valid constraints. Adding these inequalities to D leads to a so-called extended formulation. In the primal, the respective columns allow an item in a pattern to be exchanged by a combination of other items with smaller or equal total size.

Valério de Carvalho showed that any optimal solution to the non-extended formulation verifies inequalities (4.5), and hence they are weak dual-optimal inequalities. These results remain even in the case of the multiple length cutting stock problem. Extensions to the proofs are straightforward and, so, we omit them. During the successive resolutions of the restricted master problems in a column generation process, since the columns are not completely enumerated, these inequalities are frequently violated. Therefore, the inclusion of a bounded number of such cuts is opportune. Furthermore, it has the additional advantage of reducing primal degeneracy.

4.3.2 Inequalities on Rolls' Dual Variables

Let D^e (P^e) correspond to D (P, the linear programming relaxation of (3.1)-(3.4)) extended with additional inequalities on the v_k dual variables (primal columns, respectively).

$$\begin{array}{lll} (P^e) & \min\ c\lambda + f\pi & \\ & A\lambda & \geq b \\ & E\lambda + F\pi & \leq B \\ & \lambda, \pi & \geq 0 \end{array} \qquad \begin{array}{lll} (D^e) & \max\ ub + vB & \\ & uA + vE & \leq c \\ & vF & \leq f \\ & u \geq 0, v & \leq 0 \end{array}$$

A family of dual inequalities can be stated using the following argument: a cutting

pattern associated to some roll can always be reassigned to another larger roll at a cost that equals the difference between the two lengths. In the primal, this operation is allowed by including columns with a $+1$ in row $m+k$, a -1 in row $m+k'$ and a cost of $W_k - W_{k'}$, $k = 1, \ldots, K-1$, $k' = 2, \ldots, K$, and $W_k > W_{k'}$. In the dual, we will be inserting cuts with the form

$$v_k - v_{k'} \leq W_k - W_{k'},\ k = 1, \ldots, K-1,\ k' = 2, \ldots, K,\ \text{and } W_k > W_{k'}. \quad (4.6)$$

Let $(\widetilde{\lambda}, \widetilde{\pi})$ be a valid solution for P^e. In the case where $F\widetilde{\pi} \neq 0$, $\widetilde{\lambda}$ may be infeasible for P, as illustrated in Example 4.1. The next proposition and the corollary that follows show that the LP lower bounds of P and P^e remain the same.

Proposition 4.1 *Given a feasible solution $(\widetilde{\lambda}, \widetilde{\pi})$ to P^e, we can always recover a feasible solution to P with an equal or lower cost.*

Proof: Let AE and NF be the subset of columns in P^e associated to the patterns and to the dual inequalities (4.6), respectively. Starting with $(\widetilde{\lambda}, \widetilde{\pi})$, the following step-by-step procedure shows how to build a solution expressed only as a nonnegative combination of feasible cutting patterns. To clarify the presentation, the k index was dropped from the columns $(a^k_{1r}, ..., a^k_{mr}; e^k_{1r}, ..., e^k_{Kr})^T$ of P^e.

for $k := K, ..., 2$

Let $\overline{AE}$ be the subset of columns $(a_{1r}, ..., a_{mr}; e_{1r}, ..., e_{Kr})^T$ of AE with $\widetilde{\lambda}_r > 0$ and an unit coefficient in row $m+k$ ($e_{kr} = 1$).
Let $\overline{NF}$ be the subset of columns $NF_j = (0, ..., 0; ..., 1, ..., -1, ...)^T$ from NF with $\widetilde{\pi}_j > 0$, a cost of f_j units, the -1 occurring at position $m+k$ and the $+1$ at position s_j.

for all $j : NF_j \in NF$

while $\widetilde{\pi}_j > 0$ and $\sum_{r:AE_r \in \overline{AE}} \widetilde{\lambda}_r > B_k$

Select a column $(a_{1r}, ..., a_{mr}; ..., e_{s_j r}, ..., e_{kr}, ...)^T$ from $\overline{AE}$ with $\widetilde{\lambda}_r > 0$
$AE_{new} = (a_{1r}, ..., a_{mr}; ..., e_{s_j r} + 1, ..., e_{kr} - 1, ...)^T$. This column has a cost of $W_k + f_j$; λ_{new} is the associated primal variable.
if $(\widetilde{\pi}_j > \widetilde{\lambda}_r)$
$\widetilde{\pi}_j := \widetilde{\pi}_j - \widetilde{\lambda}_r$, $\lambda_{new} := \widetilde{\lambda}_r$ and $\widetilde{\lambda}_r := 0$
else
$\widetilde{\lambda}_r := \widetilde{\lambda}_r - \widetilde{\pi}_j$, $\lambda_{new} := \widetilde{\pi}_j$ and $\widetilde{\pi}_j := 0$
end if
Add AE_{new} to P^e and update the solution with λ_{new}

end while

if $\widetilde{\pi}_j > 0$, $\widetilde{\pi}_j := 0$

end for

end for

The final solution is valid for P^e, and has all the columns referring to the dual inequalities (4.6) at the zero level. Therefore, we get a feasible solution for P. If condition $\widetilde{\pi}_j > 0$ at the end of the algorithm is true at least once, the resulting solution will have a cost lower than the original one $(\widetilde{\lambda}, \widetilde{\pi})$. Otherwise, the cost remains the same. □

Corollary 1 *Since P^e is a relaxation of P, the optimal solutions of P and P^e have the same cost.*

We prove next that inequalities (4.6) are in fact weak dual-optimal inequalities.

Proposition 4.2 *Any primal-dual optimal solution pair to (P,D) satisfies inequalities (4.6).*

Proof: Consider an optimal primal-dual solution pair λ^* and (u^*, v^*) to P and D, respectively, and let $A_r^k = (a_{1r}^k, ..., a_{mr}^k)^T$ and $E_r^k = (..., 1, ...)^T$, the $+1$ occurring at position k, be the elements of a column in P with $\lambda_r^k > 0$ in λ^*. This column has a null reduced cost, *i.e.*, $W_k - u^*.A_r^k - v^*.E_r^k = 0$. A pattern with the same items reassigned to a larger roll k' is still a valid pattern. Let $A_{r'}^{k'}$ and $E_{r'}^{k'}$ be the elements of the corresponding column. We have $W_{k'} - u^*.A_{r'}^{k'} - v^*.E_{r'}^{k'} \geq 0$. Subtracting the reduced costs, we get $W_{k'} - W_k - u^*.(A_{r'}^{k'} - A_r^k) - v^*.(E_{r'}^{k'} - E_r^k) = W_{k'} - W_k - v_{k'}^* + v_k^* \geq 0$. Thus, $v_{k'}^* - v_k^* \leq W_{k'} - W_k$. The result holds no matter what roll k' is considered as long as $W_{k'} > W_k$. Consider now the case in which roll k is never used in the optimal solution λ^*. By the complementary slackness condition, v_k^* will equal 0 and, since the v variables in D are null or negative, for all the k' such that $W_{k'} > W_k$, inequality $v_{k'} - v_k \leq W_{k'} - W_k$ holds. □

Example 4.1 Consider an instance with a set W of rolls, $W = (7, 7, 5, 4, 4, 4)$, and a set w of items, $w = (3, 3, 3, 2, 2, 2, 1, 1)$. A possible restricted master problem comprising the set of linearly independent dual inequalities associated to (4.6) $(\widetilde{\pi}_1, \widetilde{\pi}_2)$ is illustrated in Figure 4.1. A solution $(\widetilde{\lambda},\widetilde{\pi})$ with $\widetilde{\lambda}_1^1$, $\widetilde{\lambda}_1^2$, $\widetilde{\lambda}_2^2$ and $\widetilde{\pi}_1$ equal to one and the other variables equal to zero is feasible for the extended model. However, $\widetilde{\lambda}$ is infeasible for the non-extended model since the limit on rolls with length 5 is exceeded. □

	λ_1^1	λ_2^1	λ_1^2	λ_2^2	λ_1^3	λ_2^3	π_1	π_2	
w_i =3	2		1		1				$\geq$ 3
2		3	1	2		2			$\geq$ 3
1	1	1		1	1				$\geq$ 2
W_k =7	1	1					1		$\leq$ 2
5			1	1			-1	1	$\leq$ 1
4					1	1		-1	$\leq$ 3
	7	7	5	5	4	4	2	1	

Figure 4.1: Restricted master problem (Example 4.1)

We can strengthen D even more using the following constraints where ε is a small positive value

$$v_K \geq W_K - W_1 - \varepsilon, \tag{4.7}$$

$$v_1 \geq \min \left\{ W_1 - \left\lceil \frac{W_1}{W_i} \right\rceil \times W_i \;\middle|\; i = 1, ..., K \right\} - \varepsilon. \tag{4.8}$$

Proposition 4.3 *Let $(\widetilde{\lambda}^*; \pi_1^*, \pi_2^*)$ be an optimal solution to $P^{e'}$, the extended version of P with the additional columns corresponding to (4.7) and (4.8). If P is feasible, then $\pi_1^* = \pi_2^* = 0$ and $\widetilde{\lambda}^*$ is optimal to P.*

Proof: The proof is made by contradiction. If π_1^* is positive, then $\sum_{r=1}^{p_K} \widetilde{\lambda}_{Kr}^* > B_K$ and $\pi_1^* = B_K - \sum_{r=1}^{p_K} \widetilde{\lambda}_{Kr}^*$. Otherwise, π_1^* can be made equal to zero; the total cost will be reduced without affecting the solution feasibility. When $\pi_1^* > 0$, a set of items is cut at the cost of $\pi_1^* W_K + \pi_1^*(W_1 - W_K + \varepsilon) = \pi_1^*(W_1 + \varepsilon)$. Since P^e is a relaxation of P and P is feasible, a solution must exist with these items reassigned to some of the other rolls. In the worst case, the patterns where these items are included remain unchanged and go to the largest roll leading to a cost of $\pi_1^* W_1$, which contradicts the fact of $(\widetilde{\lambda}^*; \pi_1^*, \pi_2^*)$ being a lower cost solution. When $\pi_2^* > 0$, some patterns are assigned to extra units of the largest roll. The respective items must fit in at least one combination of $\lceil W_1/W_i \rceil$ rolls of length W_i, $i = 1, ..., K$. The cost incurred in P^e for using an extra unit of the largest roll is greater than the cost of the worst combination where these items may fit. Therefore, if P is feasible, a lower cost solution exists and $(\widetilde{\lambda}^*; \pi_1^*, \pi_2^*)$ can not be optimal. □

With these inequalities, the dual variables are confined to a fixed and non empty box defined before starting column generation. This box strictly contains an optimal dual solution. Furthermore, instances were found with alternative optimal solutions

violating constraints (4.7) and (4.8), which indicates that these are deep dual-optimal inequalities. In the literature, there is evidence that points to a better efficiency of column generation when this kind of boxes are used [14].

Moving all the items from a roll to another empty roll leads to an increase of the unused space equal to the difference between the roll lengths, which, if it is big enough, may be fulfilled with other items. However, this operation does not translate into valid inequalities. In fact, if we consider adding items' dual variables to (4.6), the result will be a true relaxation of P with a poorer lower bound.

4.4 Extending the Dual Inequalities to the Whole Branch-and-Bound Tree

In [117], the author proved the validity of the inequalities (4.5) for the dual of the standard cutting stock problem, a model very similar to (4.1)-(4.4). Extending these proofs to the case of multiple stock lengths is immediate. The effect of the extra columns on the linear programming relaxation was studied, but no allusion was made about how they can be used within a branch-and-bound framework. In this section, we fill this gap by stating the conditions under which a cut from this family remains valid at a node of a branch-and-bound tree. Here, we consider specifically a branch-and-bound framework based on the following conditions: at each node of the tree, the LP relaxation of (3.1)-(3.4) is solved, and branching constraints are derived from an equivalent arc flow model.

As in the previous section, we will use here the terminology (P^e, D^e) to identify the primal-dual pair of problems, extended with extra columns and valid inequalities, respectively. It should be clear from the context which precise inequalities are really enforced. The primal variables for the patterns continue to be denoted by λ_i^k as in (3.1)-(3.4). Sometimes, to simplify the presentation, we will drop the k index that identifies the stock length, when it is not absolutely necessary. The extra variables in the primal for the dual inequalities will be denoted by π_i.

4.4.1 Validity of the Inequalities on Items' Dual Variables

Consider the family of dual inequalities (4.5). At a node of the branch-and-bound tree, adding the columns associated to these cuts to the LP master P may cause the violation of some branching constraints. To illustrate our assertion, we use the following simple example.

Example 4.2 Consider an instance that has a set $W = (7, 5)$ of stock lengths with availabilities $B = (5, 5)$, and a set $w = (4, 3, 2)$ of item sizes with demands $b = (2, 5, 2)$.

The complete LP master P is given in Figure 4.2.

	λ_1^1	λ_2^1	λ_3^1	λ_4^1	λ_5^1	λ_1^2	λ_2^2	λ_3^2	
w_i =4	1	1				1			$\geq$ 2
3	1		2	1			1		$\geq$ 5
2		1		2	3		1	2	$\geq$ 2
W_k =7	1	1	1	1	1				$\leq$ 5
5						1	1	1	$\leq$ 5
	7	7	7	7	7	5	5	5	

Figure 4.2: LP master (Example 4.2)

The optimum for this LP problem has a cost of 27.5 units, for a possible set of pattern frequencies of 2, 0.5 and 2 for λ_1^1, λ_3^1 and λ_2^2, respectively, and 0 for the other columns. Following the conversion scheme described in the previous chapter, we can convert this solution into the following set of arc flows: 2 units of flow in the arc $(0, 4)$, 2 units in $(4, 7)$, 2.5 units in $(0, 3)$, 0.5 units in $(3, 6)$ and 2 units in $(3, 5)$.

Assume now that the branching constraint $x_{0,3} \leq 0$ on the arc $(0, 3)$ is enforced in one of the branching nodes, say q. According to the branching scheme devised in Chapter 3, this should not be the branching constraint to enforce at this point. We use it here because it perfectly illustrates how a primal column related to a dual inequality of type (4.5) can lead to solutions that implicitly violate branching constraints. Figure 4.3 shows the resulting LP master. For this node, the optimal solution happens to be integer and has a cost of 40 units, with $\lambda_1^1 = 5$ and $\lambda_3^2 = 1$.

	λ_1^1	λ_2^1	λ_3^1	λ_4^1	λ_5^1	λ_1^2	λ_2^2	λ_3^2	
w_i =4	1	1				1			$\geq$ 2
3	1		2	1			1		$\geq$ 5
2		1		2	3		1	2	$\geq$ 2
W_k =7	1	1	1	1	1				$\leq$ 5
5						1	1	1	$\leq$ 5
$x_{0,3}$			1	1			1		$\leq$ 0
	7	7	7	7	7	5	5	5	

Figure 4.3: LP master for the branching node q (Example 4.2)

If the dual inequality $u_1 \geq u_2$ was enforced in the dual of the LP master and kept in all the nodes of the branching tree, the optimal solution in node q would not have a cost of 40 units anymore, but a cost of 29.17 units, and would be given by the following pattern frequencies: $\lambda_1^1 = 3.5$ and $\lambda_5^1 = \frac{2}{3}$. Additionally, the extra primal column would

have a value of 1.5 units. If we do not take into account the effect of the extra column for now, converting this new solution into a set of arc flows gives: $x_{0,4} = x_{4,7} = 3.5$ and $x_{0,2} = x_{2,4} = x_{4,6} = \frac{2}{3}$. Apparently, the branching constraint seems to be respected, but this is not the case. Indeed, for this set of pattern frequencies, what the extra column does in practice by having a positive value is to replace an item with a size of 4 units by another item with 3 units of size, in the pattern related to λ_1^1. The resulting pattern is the same as the one associated to λ_3^1, and hence, we do have a positive flow in the arc $(0, 3)$.

For the branching node q, the model with the extra column is a true relaxation of P. The dual interpretation is that the inequality is not valid for the dual problem D. □

We can summarize as follows the reason why some branching constraints may be violated when dual inequalities of type (4.5) are considered. In the primal, these cuts translate into columns that allow an item in a pattern to be interchanged with other items, which have a smaller or equal total size. The problem is that we can not restrict the set of patterns from which these operations take place. Hence, combining a pattern P_i with a column related to a dual cut of type (4.5) can result in a pattern P_j with one of the following characteristics:

. pattern P_j has a $+1$ coefficient in the row of a branching constraint imposed on an arc (s, t), but the corresponding path in the graph G of the arc flow model obtained with our conversion scheme does not include this arc;

. pattern P_j has a null coefficient in the branching constraint on an arc (s, t), when the corresponding path in G does in fact include this arc (this is the case that arises in Example 4.2).

If pattern P_i and the primal column for some dual inequality of type (4.5) have positive values, the flow over (s, t) might not be correctly accounted. Consequently, while a branching constraint on (s, t) seems to be respected, in practice it might not be.

4.4.2 Validity Conditions

Some of the inequalities (4.5) may remain valid, depending on the specific set of arcs on which a branching constraint has been enforced. The following Proposition states the sufficient conditions that guarantee the validity of these dual cuts at a node q of the branch-and-bound tree.

Proposition 4.4 *For all the arcs (s, t) of G on which at least one branching constraint has been enforced at node q, and for an item i and a subset S of the item sizes such*

that $w_i \geq \sum_{l \in S} w_l$*, if at least one of the following conditions applies*

$$t - s > w_i, \tag{4.9}$$

$$\sum_{\{l \in S : w_l > t-s\}} w_l > s, \tag{4.10}$$

then $u_i \geq \sum_{l \in S} u_l$ *will be a valid dual-optimal inequality at node* q*.*

Proof: Let $(\widetilde{\lambda}, \widetilde{\pi})$ be a feasible solution to the extended primal problem P^e, resulting from P plus the columns for the dual inequalities (4.5) that satisfy condition (4.9), (4.10) or both. We can show that an optimal solution to P^e can be mapped into an equal cost solution to P composed by a set of columns that are feasible for P. Let n be the number of branching constraints enforced at the node q, AER the set of columns for the patterns in P (A represents the pattern's coefficients, E the coefficients in the rolls' availability constraints and R the coefficients in the branching constraints), F the columns associated to the dual inequalities and J^q the set of arcs with a branching constraint at the node q. To perform the mapping, we follow the step-by-step procedure described below.

for $i := 1, ..., m-1$

Let $\overline{AER}$ be the subset of columns $(a_{1p}, a_{2p}, ..., a_{mp}; e_{1p}, ..., e_{Kp}; r_{1p}, ..., r_{np})^T$ in the extended problem P^e, with a positive value $\widetilde{\lambda}_p > 0$ and such that $a_{ip} > 0$.
Let $\overline{F}$ be the subset of columns $(..., -1, ..., 1, ..., 1, ...; ..., 0, ...; ..., 0, ...)^T$ that corresponds to the dual cuts with a positive value in $\widetilde{\pi}$ and a -1 in row i. For a column $\overline{F_j}$, the elements $+1$ occur at the positions s_j^1,...,$s_j^{|S|}$.

for all $j : \overline{F}_j \in \overline{F}$

$p := 1$

while $p \leq |\overline{AER}|$ and $\widetilde{\pi}_j > 0$

$\overline{AER}_p = (..., a_{ip}, ..., a_{s_j^1 p}, ..., a_{s_j^{|S|} p}, ...; e_{1p}, ..., e_{Kp}; r_{1p}, ..., r_{np})^T$

if $a_{ip} > 0$

$AER_{new} = (..., 0, ..., a_{s_j^1 p} + a_{ip}, ..., a_{s_j^{|S|} p} + a_{ip}, ...; e_{1p}, ..., e_{Kp}; r_{1p}, ..., r_{np})^T$

Let λ_{new} be the primal variable related to AER_{new}.

The cost of AER_{new} remains equal to the cost of $\overline{AER}_p$.

For an arc $(s,t) \in J^q$, if (s,t) belongs to the path in G for $\overline{AER}_p$, then for all the indexes d of the related branching constraints, we will have $r_{dp} = 1$, otherwise $r_{dp} = 0$.

For an arc $(s,t) \in J^q$ and $\overline{F}_j$, if condition (4.9) applies, two situations may arise: (s,t) belongs (does not belong) to the path related

to $\overline{AER}_p$, and in this case it will (will not) belong to the path for AER_{new}, since the removal of the item i from $\overline{AER}_p$ and addition of the items of S affects only the arcs of $\overline{AER}_p$ at the right of (s,t). For an arc $(s,t) \in J^q$ and $\overline{F}_j$, if (4.10) applies, then (s,t) can not belong to $\overline{AER}_p$. Since $s \geq 0$, at least one item l of S is such that $w_l > t - s$, otherwise (4.10) would not be satisfied. Consequently, $w_i > t - s$ and the arc for item i, say (s_i, t_i), appears in the path for AER_{new} at the left of the arcs for the items with size $t - s$. Even if $s_i = 0$, no arc will never begin at position s. Since $\sum_{\{l \in S: w_l > t-s\}} w_l > s$, the path for AER_{new} will have no arc beginning at position s.

if $(\widetilde{\pi}_j > a_{ip}\widetilde{\lambda}_p)$

$\widetilde{\pi}_j := \widetilde{\pi}_j - a_{ip}\widetilde{\lambda}_p$, $\lambda_{new} := \widetilde{\lambda}_p$ and $\widetilde{\lambda}_p := 0$

else

$\widetilde{\lambda}_p := \widetilde{\lambda}_p - \widetilde{\pi}_j / a_{ip}$, $\lambda_{new} := \widetilde{\pi}_j / a_{ip}$ and $\widetilde{\pi}_j := 0$

end if

Add AER_{new} to P^e with λ_{new}.

end if

$p := p + 1$

end while

end for

end for

At the end of the procedure, all the primal variables for the dual inequalities will have a null value. The new solution satisfies all the constraints of the problem, and is composed by valid columns, *i.e.* columns with correct coefficients in the rows associated to the branching constraints. All the operations are done at no cost, and, so, the solution is also optimal for the original primal problem P. With dual inequalities satisfying one of the conditions stated above, the feasible solution space for D^e still includes at least one optimal dual solution to D, and hence these inequalities are dual-optimal at the branching node q. □

4.4.3 New Dual-Optimal Inequalities

In the nodes of the branching tree, a new family of valid dual inequalities can be used relating the dual variables for the demand constraints with the ones for the branching restrictions. These inequalities are presented in the following Proposition.

Proposition 4.5 *For a branching node q, let H^q and G^q be respectively the set of "greater than or equal to" and "less than or equal to" branching constraints. The dual variables for the l^{th} constraint related to H^q and G^q are denoted respectively by $h^l_{(s,t)}$ and $g^l_{(s,t)}$, with (s,t) identifying the arc on which the constraint is enforced. For the branching node q, and for a single item i and subset of item sizes S such that $w_i \geq \sum_{l \in S} w_l$, the following dual inequalities*

$$u_i + \sum_{\{l \in H^q : t-s \leq w_i\}} h^l_{s,t} \geq \sum_{l \in S} u_l + \sum_{\{l \in G^q : t-s \leq w_i\}} g^l_{s,t}, \ i = 1, ..., m, \ \forall S. \quad (4.11)$$

are valid dual-optimal inequalities for the dual problem D.

Proof: The proof consists essentially in a sequence of steps similar to the ones used in the proof of Proposition 4.4. The column for the dual inequality denoted by $\overline{F}_j$ in Proposition 4.4 has now the form

$$(..., -1, ..., 1, ..., 1, ...; ..., 0, ...; ..., -1, ..., -1, ..., 1, ..., 1, ...)^T,$$

i.e. this column has non-negative coefficients in some of the rows corresponding to the branching constraints. The elements -1 in the rows for these constraints occur at positions $m+K+y^1_j$, $m+K+y^2_j$, ..., $m+K+y^{|Y|}_j$, with $Y = \{l : \ l \in H^q \ \texttt{and} \ t-s \leq w_i\}$. In turn, the elements $+1$ in the rows for the branching constraints are in positions $m + K + z^1_j$, $m + K + z^2_j$, ..., $m + K + z^{|Z|}_j$, with $Z = \{l : \ l \in G^q \ \texttt{and} \ t - s \leq w_i\}$. The new pattern AER_{new} takes the form

$$\begin{aligned}(..., 0, ..., a_{s^1_j p} + a_{ip}, ..., a_{s^{|S|}_j p} + a_{ip}, ...; e_{1p}, ..., e_{Kp}; \\ ..., r_{y^1_j p} - a_{ip}, ..., r_{y^{|Y|}_j p} - a_{ip}, \\ ..., r_{z^1_j p} + a_{ip}, ..., r_{z^{|Z|}_j p} + a_{ip})^T,\end{aligned}$$

In AER_{new}, if for example $r_{y^l_j p} - a_{ip} < 0$, since this element can only occur in the row for a "greater than or equal to" branching constraint, we can set it to zero. The resulting solution with the updated column AER_{new} will continue to satisfy the branching constraint y^l_j. The same can be done to all the negative elements of AER_{new}.

Similarly, if $r_{z^l_j p} + a_{ip} > 1$, since such an element can only occur in the rows for "less than or equal to" branching constraints, setting it to 1 will result in a solution that still satisfies the branching constraint z^l_j.

Finally, if $r_{z^l_j p} + a_{ip} > 1$, but the arc associated to the branching constraint z^l_j does not belong to the path in G for AER_{new}, this element is set to 0. Again, this situation only arises in rows for "less than or equal to" constraints, and, so, this operation does not affect the feasibility of the new solution.

In all cases, we are always able to recover from AER_{new} a column valid in P without changing its cost, nor the feasibility of the whole solution. □

4.5 Computational Experiments

To evaluate the impact of using dual-optimal inequalities in the whole branch-and-bound tree, we repeated part of the tests described in Section 3.6 with the randomly generated instances, and the set of 50 instances tested in [13]. The experiments were conducted in the same conditions.

Tables 4.1, 4.2, 4.3 and 4.4 illustrate the computational results obtained when dual inequalities are applied in all the branching nodes. The columns have the same meaning as in Section 3.6, except for $cols_{IN}$ that now represents the number of initial columns in the master that are patterns or columns related to dual inequalities.

On average, the results obtained when dual inequalities are extended to the whole branch-and-bound tree are significantly better. Besides the reduction in the number of pricing subproblems solved at the root node and the number of generated columns, there is also a drastic reduction in the number of branching nodes explored. For the group of instances vs3, for example, an average of 7.55 nodes were necessary, versus 28.70 without dual cuts. For the instances tested in [13], the reduction of branching nodes amounts to almost 70%, while the computing time is divided by two. For these instances, we were also able to solve 94% of the instances tested in [13], while only 90% were solved when dual inequalities were not used.

4.6 Alternative Aggregation Schemes

In the past, aggregation and disaggregation techniques were essentially used to deal with problems coming from memory space restrictions. These difficulties were becoming even more constraining as researchers were trying to solve problems with a growing level of precision. By concentrating on a restricted but sufficiently representative set of data, one will also expect to see the computational burden reduced. Rogers *et al.* [96] gave a comprehensive survey on the contributions made in the field and synthesized the different elements of an aggregation/disaggregation process.

In this section, we describe two alternative aggregation schemes. The goal is to accelerate the search for an optimal solution through column generation by solving sequences of easier approximations, *i.e.*, aggregated problems. Here, aggregation is seen as an implicit form of controlling the dual variables.

The methods rely exclusively on item aggregation and, therefore, they can be applied to the standard cutting stock problem. In Section 4.6.1, we develop a simple two-phase column generation algorithm that starts by solving a row aggregated model that corresponds to a problem with larger items. In Section 4.6.2, an iterative algorithm is presented that takes advantage of the phenomenon of identical prices, early pointed out by Gilmore and Gomory [53], using smaller LPs where both variables and

$Name$	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	z_{wb}	z_{LP}	z^*	z_{WB}
vs300	17	9	37	10	61	0	0	0	0.00	0.01	0.00	0.01	1269	1276.42	1371.00	1371
vs301	17	9	36	12	82	107	107	73	0.02	0.02	0.15	0.19	1030	1030.00	1031.00	1450
vs302	16	11	40	7	57	0	0	0	0.00	0.02	0.00	0.02	1256	1271.62	1353.00	1353
vs303	19	11	41	11	85	1	0	1	0.01	0.03	0.00	0.04	1068	1068.00	1068.00	1289
vs304	15	11	32	5	40	13	31	7	0.00	0.01	0.01	0.02	1142	1144.13	1177.00	1318
vs305	19	10	41	8	68	0	0	0	0.02	0.02	0.00	0.04	1164	1171.00	1171.00	1283
vs306	17	11	34	9	85	6	21	4	0.02	0.04	0.00	0.06	1245	1248.88	1259.00	1408
vs307	18	11	38	7	64	1	0	1	0.00	0.02	0.00	0.02	1049	1049.00	1071.00	1351
vs308	18	9	39	9	66	12	19	8	0.00	0.02	0.00	0.02	1176	1179.44	1182.00	1386
vs309	17	9	40	7	42	0	0	0	0.00	0.01	0.00	0.01	1280	1294.25	1385.00	1385
vs310	19	9	38	8	54	0	0	0	0.00	0.01	0.00	0.01	1182	1183.94	1203.00	1350
vs311	18	10	39	15	94	46	25	38	0.00	0.02	0.06	0.08	1242	1247.55	1258.00	1358
vs312	20	10	42	9	77	14	30	9	0.02	0.02	0.05	0.09	1199	1199.25	1222.00	1367
vs313	20	11	43	8	77	3	13	1	0.00	0.03	0.00	0.03	1173	1173.40	1178.00	1312
vs314	17	9	38	11	68	7	8	6	0.00	0.03	0.00	0.03	1201	1208.67	1214.00	1342
vs315	19	8	42	10	62	0	0	0	0.00	0.02	0.00	0.02	1298	1344.00	1344.00	1344
vs316	19	11	41	8	75	5	13	2	0.00	0.03	0.00	0.03	1094	1094.00	1095.00	1354
vs317	17	11	39	7	61	0	0	0	0.00	0.02	0.00	0.02	1275	1391.00	1391.00	1391
vs318	18	10	40	11	86	0	0	0	0.01	0.03	0.00	0.04	1273	1307.94	1314.00	1314
vs319	18	10	40	8	68	2	10	1	0.00	0.02	0.00	0.02	1229	1234.00	1243.00	1358
avg.	17.90	10.00	39.00	9.00	68.60	10.85	13.85	7.55	0.01	0.02	0.01	0.02	1192.25	1205.82	1226.50	1354.20

Table 4.1: Computational results with dual inequalities (vs3)

$Name$	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	z_{wb}	z_{LP}	z^*	z_{WB}
vs500	26	14	60	10	117	1	0	1	0.01	0.02	0.00	0.03	1869	1869.96	1891.00	2039
vs501	25	12	58	8	68	0	0	0	0.02	0.01	0.00	0.03	1833	1837.07	1924.00	1924
vs502	25	14	56	11	126	6	30	2	0.01	0.02	0.00	0.03	1668	1668.00	1668.00	1883
vs503	27	13	58	8	88	42	96	28	0.02	0.01	0.28	0.31	1626	1626.00	1628.00	1940
vs504	23	15	47	10	118	225	139	169	0.02	0.01	0.76	0.79	1798	1798.00	1798.00	2042
vs505	26	13	60	6	64	4	13	3	0.02	0.02	0.00	0.04	1734	1734.85	1753.00	1900
vs506	24	13	57	8	89	0	0	0	0.01	0.01	0.00	0.02	1830	1834.30	1889.00	2037
vs507	25	15	58	7	88	5	29	3	0.02	0.02	0.00	0.04	1792	1792.00	1831.00	1975
vs508	29	15	63	10	132	37	82	29	0.02	0.02	0.22	0.26	1700	1700.00	1704.00	2024
vs509	24	15	59	8	100	0	0	0	0.02	0.02	0.00	0.04	1885	1891.13	1958.00	1958
vs510	28	12	65	8	73	0	0	0	0.01	0.02	0.00	0.03	1872	1880.67	1904.00	1904
vs511	26	16	61	14	175	123	92	86	0.02	0.03	0.45	0.50	1832	1832.65	1836.00	1963
vs512	24	14	53	5	54	0	0	0	0.02	0.01	0.00	0.03	1865	1872.38	1938.00	1938
vs513	26	13	58	10	109	19	17	17	0.02	0.02	0.09	0.13	1759	1759.00	1760.00	1994
vs514	27	14	61	7	82	2	14	1	0.03	0.02	0.00	0.05	1781	1781.00	1804.00	1952
vs515	25	12	55	9	93	58	93	46	0.01	0.02	0.25	0.28	1716	1716.00	1718.00	1966
vs517	26	13	57	5	49	0	0	0	0.02	0.01	0.00	0.03	1882	1884.35	1945.00	1945
vs518	26	15	59	10	131	95	188	60	0.02	0.02	0.63	0.67	1644	1644.00	1644.00	1886
vs519	27	15	64	10	123	2	13	1	0.02	0.02	0.00	0.04	1825	1825.00	1825.00	1941
avg.	25.74	13.84	58.37	8.63	98.89	32.58	42.42	23.47	0.02	0.02	0.14	0.18	1784.79	1786.65	1811.47	1958.47

Table 4.2: Computational results with dual inequalities (vs5)

$Name$	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}	z_{wb}	z_{LP}	z^*	z_{WB}
vs600	18	14	40	11	102	59	30	45	0.02	0.01	0.13	0.16	1279	1280.80	1282.00	2039
vs601	19	12	37	11	98	0	0	0	0.01	0.01	0.00	0.02	1115	1115.00	1115.00	1934
vs602	17	12	35	10	94	190	144	146	0.02	0.03	0.39	0.44	1031	1031.00	1032.00	2045
vs603	18	15	41	8	95	28	48	16	0.02	0.01	0.08	0.11	1115	1115.00	1115.00	1935
vs604	19	12	41	8	77	12	29	5	0.00	0.01	0.03	0.04	1138	1138.00	1138.00	1901
vs605	17	14	36	9	99	11	15	6	0.00	0.02	0.00	0.02	1274	1276.00	1277.00	2009
vs606	18	16	39	7	96	2	16	1	0.00	0.01	0.00	0.01	1084	1084.00	1084.00	1983
vs607	19	13	39	8	84	2	6	1	0.01	0.01	0.00	0.02	1128	1130.00	1130.00	1991
vs608	17	15	39	12	108	8	9	5	0.01	0.02	0.00	0.03	1223	1224.56	1225.00	1996
vs609	17	14	40	12	129	68	108	44	0.02	0.01	0.18	0.21	1169	1172.00	1172.00	2019
vs610	19	14	41	8	86	12	18	8	0.01	0.02	0.02	0.05	1177	1177.00	1177.00	2045
vs611	19	15	40	7	87	12	37	6	0.00	0.01	0.02	0.03	1082	1082.00	1082.00	1958
vs612	17	12	38	13	103	77	73	54	0.01	0.02	0.12	0.15	1216	1216.90	1220.00	1971
vs613	19	12	45	10	62	4	10	2	0.01	0.02	0.00	0.03	1262	1263.17	1264.00	2016
vs614	19	15	43	11	113	45	74	27	0.00	0.02	0.08	0.10	1114	1114.00	1114.00	1938
vs615	15	14	36	8	92	5	19	3	0.00	0.01	0.00	0.01	1071	1071.00	1071.00	2083
vs616	17	13	39	12	93	38	44	26	0.00	0.02	0.08	0.10	1206	1208.94	1212.00	2016
vs617	17	14	35	7	76	6	13	3	0.00	0.02	0.01	0.03	1137	1137.00	1137.00	2041
vs618	20	15	43	10	104	23	17	15	0.03	0.02	0.01	0.06	1258	1260.00	1260.00	1955
vs619	18	13	42	11	87	4	3	2	0.01	0.02	0.00	0.03	1209	1212.00	1213.00	1908
avg.	17.95	13.70	39.45	9.65	94.25	30.30	35.65	20.75	0.01	0.02	0.06	0.08	1164.40	1165.42	1166.00	1989.15

Table 4.3: Computational results with dual inequalities (vs6)

$Inst$	m	K	$cols_{IN}$	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	t_{PP}	t_{LP}	t_{BB}	t_{TOT}
2	100	3	297	88	220	3	0	3	0.97	2.96	2.73	6.66
4	99	4	311	99	303	1	0	1	1.13	1.41	0.63	3.17
5	100	4	298	111	363	0	0	0	0.97	1.33	0.00	2.30
6	99	4	299	124	413	7	1	6	0.95	1.47	9.71	12.13
7	99	4	300	107	325	7	0	7	1.13	1.20	6.66	8.99
8	99	3	300	80	201	8	5	6	1.22	0.41	8.50	10.13
9	100	4	309	108	346	25	5	21	0.91	1.19	29.78	31.88
10	99	4	303	98	344	0	0	0	1.00	1.28	0.00	2.28
11	98	4	293	99	324	0	0	0	1.13	1.06	0.00	2.19
12	98	4	314	118	339	26	11	22	1.41	1.44	29.64	32.49
14	99	4	305	107	339	7	0	7	1.17	1.48	8.69	11.34
15	100	4	307	112	393	6	1	5	0.88	1.92	6.99	9.79
16	100	3	307	155	366	26	3	24	1.28	6.49	39.79	47.56
17	100	2	315	142	282	15	2	14	1.13	8.78	15.73	25.64
18	100	3	298	117	314	3	0	3	0.97	5.99	2.72	9.68
19	99	4	297	79	250	3	0	3	1.14	0.64	2.02	3.80
20	99	4	307	125	419	1	0	1	1.09	2.16	0.53	3.78
21	98	4	297	86	300	8	0	8	0.81	1.45	10.42	12.68
22	99	3	299	100	256	4	0	4	1.03	0.81	4.22	6.06
23	98	4	301	102	334	5	2	3	0.94	2.36	4.76	8.06
24	99	4	302	52	192	0	0	0	1.06	0.30	0.00	1.36
25	100	4	309	129	390	23	1	22	1.23	1.39	25.85	28.47
26	100	3	303	130	356	9	0	9	0.75	2.23	11.92	14.90
27	99	4	303	75	258	0	0	0	0.94	0.95	0.00	1.89
28	98	3	297	93	247	4	0	4	0.73	1.34	5.25	7.32
29	99	4	315	120	418	7	0	7	0.94	3.34	9.79	14.07
30	99	4	312	104	348	3	0	3	1.24	1.95	3.45	6.64
31	99	4	301	81	270	25	3	22	0.70	1.13	23.88	25.71
32	99	4	301	90	279	10	4	8	1.14	1.03	16.50	18.67
33	98	4	306	76	242	16	1	16	1.20	0.62	11.11	12.93
34	100	4	308	115	375	17	8	13	1.39	2.41	26.14	29.94
35	100	4	306	172	537	5	0	5	1.08	7.92	9.83	18.83
36	100	4	313	95	350	8	3	6	1.13	2.25	9.38	12.76
37	100	4	307	99	333	3	0	3	1.09	1.42	2.92	5.43
38	100	4	316	71	261	0	0	0	1.13	0.89	0.00	2.02
39	98	4	300	101	350	0	0	0	0.89	1.07	0.00	1.96
40	99	4	302	95	307	1	0	1	1.27	1.38	1.44	4.09
41	100	4	308	130	377	81	30	65	1.16	1.56	122.29	125.01
42	98	3	291	184	318	0	0	0	1.72	1.39	0.00	3.11
43	99	4	302	111	253	0	0	0	1.22	0.81	0.00	2.03
44	99	4	301	74	243	12	5	10	1.22	1.00	17.03	19.25
45	100	4	305	72	227	3	7	2	1.25	0.19	1.72	3.16
46	99	2	292	30	46	11	4	9	1.05	0.03	14.48	15.56
47	98	4	292	81	256	4	1	3	0.95	0.75	4.08	5.78
48	98	4	296	153	449	36	2	34	0.97	2.72	52.81	56.50
49	99	4	299	89	298	16	3	15	1.08	1.30	24.52	26.90
50	99	4	303	129	434	1	0	1	0.94	3.11	1.84	5.89
avg.	99.13	3.74	303.13	104.43	315.85	9.57	2.17	8.43	1.08	1.92	12.34	15.34

Table 4.4: Computational results with dual inequalities for the instances in [13]

constraints have been aggregated. The resulting doubly aggregated models give a good approximation to the disaggregated problems.

4.6.1 A Simple Row Aggregation Scheme

When the number of items per roll in a multiple length cutting stock problem is high, the resulting high density LP matrix favors the occurrence of primal degeneracy. If we were able to anticipate some of the item combinations of an optimal solution, larger items could be defined and density could be reduced. However, finding the right combination is difficult to achieve, and, so, we will approach this problem heuristically.

We solve the multiple length cutting stock problem by column generation in two steps. In the first, the items of the original problem are combined pairwise, leading to an approximation P^{ra} of P. Since we are restricting the original solution space, we get an upper bound to the optimal solution of P; its quality will be as good as our guess for the item combination. Note that the quality of the approximation will certainly tend to decrease if greater combinations are tried. In the second step, we guarantee the convergence to the optimal solution of P by solving P starting with the columns of the last restricted master problem relative to P^{ra}, properly disaggregated. We use the simple scheme RA to aggregate the items of the original problem.

Aggregation scheme RA
Let w^{ra} and b^{ra} be respectively the set of item sizes and demands of the aggregated problem. Initially, $w^{ra} = \varnothing$ and $b^{ra} = \varnothing$.
Let b, indexed by i, represent the set of demands of the original problem and q be the number of positive values in b.
$i := 1$
while $i \leq m$ and $q \geq 2$

while $b_i > 0$ and $q \geq 2$

Choose an item of the original problem with size w_j, $j = i+1, ..., m$, and $b_j > 0$ such that:
$min\{W_k - (w_i + w_j) \mid W_k - (w_i + w_j) > 0, k = 1, ..., K\} \leq W_{k'} - (w_i + w_l)$,
$\forall\ k' = 1, ..., K$, $l = i+1, ..., m$, $b_l > 0$ and $W_{k'} - (w_i + w_l) > 0$

if $(b_j \geq b_i)$

$b_j := b_j - b_i$, $b^{ra}_{new} := b_i$ and $b_i := 0$

else

$b_i := b_i - b_j$, $b^{ra}_{new} := b_j$ and $b_j := 0$

end if

Add size $w_i + w_j$ to w^{ra} with a demand of b^{ra}_{new} units, if it does not already exist, or increment by b^{ra}_{new} its demand, otherwise.

$i := i + 1$

end while

end while

if $b_i > 0$, add w_i to w^{ra} with a demand of b_i units if it does not already exist or increment by b_i its demand, otherwise.

Clearly, the total number of items ($\sum_{i=1}^{|w^{ra}|} b^{ra}_i$) decreases, while the number of different types of items in the aggregated problem will only be smaller compared to the original problem if equality $b_j = b_i$ holds or if b_i is equal to 0 in the last step. Since the combination of items with the same size is not allowed, the number of different types of items will never increase.

An item is combined with a smaller one if the difference between the sum of their sizes and the length of the smallest roll where this combination fits is the minimum over all the possible combinations that include the first item. With such a criterion, if an item i goes with an item j, item $i + 1$ will frequently be combined with the item $j - 1$, particularly if $w_i + w_j$ is almost equal to $w_{i+1} + w_{j-1}$. This type of association is frequent in optimal solutions, where some of the patterns differ only in a small number of items. The unused space within a pattern is used to cope with the small difference between pairs of items. These pairs finally appear in very similar patterns. Another interesting point is the fact that the dual inequalities of Section 4.3 are particularly efficient in terms of stabilization for instances with groups of almost identical items.

The rows of P are subject to an equivalent aggregation process. The following example illustrates the steps followed by the aggregation scheme.

Example 4.3 Consider an instance with a set of item sizes $w = (90, 59, 58, 57, 25, 20)$ and demands $b = (5, 5, 7, 3, 7, 1)$. Ten rolls with lengths 160, 120 and 100 are available. Figure 4.4 illustrates a possible restricted master problem. The aggregation proceeds as follows. We start with $w^{ra} = \emptyset$ and $b^{ra} = \emptyset$. Five units of $w_1 = 90$ are initially aggregated to 5 other units of w_5=25, leading to $w^{ra} = (115)$ and $b^{ra} = (5)$. The 5 units left in $W_2 = 120$, the smallest roll where an item of size 115 fits, are the smallest unused space over all the other combinations that include w_1. Subsequently, 5 items of size 59 are combined 5 items of size 58 ($w^{ra} = (117, 115)$, $b^{ra} = (5, 5)$), 2 items of size 58 with 2 items of size 57 ($w^{ra} = (127, 115)$, $b^{ra} = (5, 7)$), 1 item of size 57 with 1 item of size 25 ($w^{ra} = (117, 115, 82)$, $b^{ra} = (5, 7, 1)$), and finally 1 item of size 25 with 1 item of size 20 ($w^{ra} = (117, 115, 82, 45)$, $b^{ra} = (5, 7, 1, 1)$). The aggregated problem has 4 different types of items for a total demand of 14 units, corresponding to half the 28 items of the original problem.

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9	λ_{10}	λ_{11}	λ_{12}	λ_{13}	
w_i =90	1	1						1						$\geq$ 5
59			2		1	1			1		1			$\geq$ 5
58		1		2	1		1		1	1				$\geq$ 7
57						1	1			1	1	1		$\geq$ 3
25	2		1	1	1	1	1	1				1	2	$\geq$ 7
20	1						1						2	$\geq$ 1
W_k =160	1	1	1	1	1	1	1							$\leq$ 10
120								1	1	1	1			$\leq$ 10
100												1	1	$\leq$ 10

Figure 4.4: Restricted master problem (Example 4.3)

	λ_1^{ra}	λ_8^{ra}	λ_9^{ra}	λ_{12}^{ra}	λ_{13}^{ra}	
w_i =117			1			$\geq$ 5
115	1	1				$\geq$ 7
82				1		$\geq$ 1
45	1				2	$\geq$ 1
W_k =160	1					$\leq$ 10
120		1	1			$\leq$ 10
100				1	1	$\leq$ 10

Figure 4.5: Row aggregated LP (Example 4.3)

The item aggregation leads to a similar row aggregation in P. Figure 4.5 shows the result for the restricted master problem of Figure 4.4. A column in P will have an associated column in the aggregated model only if all the items of the corresponding pattern can be combined among them according to the item aggregation defined by RA. Thus, the first column in Figure 4.4 will have a counterpart in the aggregated LP, while the second will not. For the latter, the unique item of size 90 combines with the item of size 25 and the remaining item of size 25 goes with the item of size 20. The disaggregation process is straightforward. Note that a column of the aggregated formulation may generate one or more columns for the disaggregated problem. □

Besides the sparser density of the aggregated problem, which reduces its degeneracy, this problem is also easier to solve for other obvious reasons. The set of valid cutting patterns is smaller compared to the original problem. Fewer items are needed to fulfill the rolls and their demands are also lower. The column generator subproblems are then solved faster. Moreover, the disaggregation cost is not high. The effectiveness of the

algorithm depends on the quality of the item combination. We are looking for a good set of initial columns to start the resolution of the original problem by column generation, and we expect that the effort to find them will be essentially concentrated in the first phase, where a simpler and less degenerated problem is solved. The computational experiments presented in Section 4.7 show that for some instances this algorithm brings some appreciable improvements.

4.6.2 Implicit Dual Constraints: a Double Aggregation Scheme

The optimal solutions of cutting stock problems have an interesting characteristic: items that are almost identical have dual variables that are frequently equal. Gilmore and Gomory ([53]) already experienced this phenomenon of identical prices in various stages of the column generation algorithm, for low and high waste problems. They give an explanation based on the fact that the unused space in some cutting patterns belonging to the basis allows some of their items to be interchanged.

In formulations where a demand constraint is defined for each single item, there exists an optimal solution with equal dual variables for the items of the same size. This result is well known. Ben Amor *et al.* ([15]) report computational results where it is shown that convergence is improved when a unique constraint per item size is considered. What happens is that the dual variables of identical items are forced to be equal. In this section, we propose an algorithm that solves successive approximations obtained by imposing these equality constraints to items with different sizes. From the problem definition standpoint, items with nearly the same size are replaced by a single item with a size equal to the average size of the original items. This operation amounts to aggregating columns followed by a constraint aggregation. A column of the aggregated model will correspond to a set of columns with more rows in the disaggregated model. The following example illustrates this correspondence.

Example 4.4 Remember the problem of Example 4.3. Patterns $(0,1,1,0,0,0;0,1,0)^T$, $(0,1,0,1,0,0;0,1,0)^T$ and $(0,0,1,1,0,0;0,1,0)^T$ of λ_9, λ_{10} and λ_{11}, respectively, differ in a single item. There are other patterns that differ from the previous in a single item such as $(0,2,0,0,0,0;0,1,0)^T$, $(0,0,2,0,0,0;0,1,0)^T$ and $(0,0,0,2,0,0;0,1,0)^T$. If we replace the items of size 59, 58 and 57 by an item with size 58 (the average of 59, 58 and 57) and a demand of 15 units, the pattern $(0,2,0,0;0,1,0)^T$ in the aggregated model will now stand for all the aforementioned patterns. □

The aggregation scheme is now described more formally. We define an equivalence group G_l as the set of successive items $i, i+1, ..., i+|G_l|$ such that $w_i - w_{i+1} \leq \Delta$. Let m^{da} be the total number of equivalence groups in the original problem for some positive value Δ. We have $m^{da} \leq m$. Two patterns AE_1 and AE_2 are said to be adjacent if

they relate to the same roll length and if AE_1 is obtained from AE_2, and vice versa, by replacing some items of AE_2 each one with an item belonging to the same equivalence group. The columns of P, the original formulation, are divided in groups of adjacent patterns, say $\overline{AE}_r$, $r = 1, ..., \overline{p}$. The aggregated model P^{da} results from the aggregation of columns and rows within each set $\overline{AE}_r$ defined by the following linear operations

$$T \; . \; \overline{AE}_r \; . \; S_r,$$

where T is a $m^{da} \times m$ block triangular matrix and S_r is a $|\overline{AE}_r| \times 1$ column vector, $r = 1, ..., \overline{p}$. There is a unit coefficient in row l and column i of T iff $i \in G_l$. All the elements of S_r are equal to $1/|\overline{AE}_r|$.

The question now is which size should have the m^{da} items of the aggregated problem, one per each original equivalence group, such that all the columns in P^{da} (and no others) are feasible patterns. According to the aggregation scheme, these item sizes depend on each column of P^{da} and, in particular, on the original set of adjacent columns $\overline{AE}_r$ in P that originate them. A way to have a unique size for each of the m^{da} items is to make the equivalence groups coincide with the set of dual variables that are equal in an optimal dual solution and to set the sizes equal to the optimal dual values accordingly. However, an optimal dual solution is rarely available beforehand. Searching for a set of sizes such that all the columns in P^{da} are feasible starting with a set of predefined equivalence groups is not guaranteed to succeed. In fact, such sizes may not exist. The alternative consists in using approximate sizes. Setting the size of i^{th} item, $i = 1, ..., m^{da}$, equal to the size of the smallest item in G_i leads to an aggregated problem that is a relaxation of P. On the other hand, if we choose the size of the largest item in G_i, P^{da} will be a restriction of P. We chose to use the first integer value lower than the average size of the items in G_i, $i = 1, ..., m^{da}$. The consequence is that some columns of P may not have a representation in the resulting P^{da}.

Example 4.5 Consider an instance with a set of items $w = (80, 70, 60, 55, 43, 38, 33, 20, 16)$ to be cut from rolls of length 100. Figure 4.6 shows the complete set of feasible patterns for this instance. With $\Delta = 5$, five equivalence groups can be defined: $G_1 = (80)$, $G_2 = (70)$, $G_3 = (60, 55)$, $G_4 = (43, 38, 33)$ and $G_5 = (20, 16)$. They lead to the 7 different sets of adjacent columns identified in Figure 4.6.
Figure 4.7 and 4.8 illustrate the result of the successive column and row aggregation, respectively. The new set of items for the aggregated problem is $w^{da} = (80, 70, 57, 38, 18)$. For the pattern $(0, 0, 1, 1, 0)^T$, which replaces the columns of $\overline{AE}_1$, the size of the item that stands for those of G_4 should be 37 $((1/5 \times 43 + 2/5 \times 38 + 2/5 \times 33)/(1/5 + 2/5 + 2/5))$, while for the pattern $(0, 0, 0, 2, 1)^T$ this size should be 36.6 $((3/9 \times 43 + 7/9 \times 38 + 8/9 \times 33)/2)$.
By using the averages for each equivalence group, pattern $(0, 0, 0, 0, 0, 0, 3, 0, 0)^T$ can

no longer be represented in the aggregated model ($3 \times 38 > 100$) and the aggregation of $(0, 0, 0, 0, 2, 0, 0, 0, 0)^T$ results in a pattern $(0, 0, 0, 2, 0)^T$ that has enough waste to cope with another item. Note that it is convenient for the items of an equivalence group to appear in quite similar patterns. This situation is likely to appear when the item sizes are relatively near. Imagine for example including the items of size 80 and 70 in the same equivalence group. Items of size 66 cannot combine with items of size 38, and, hence, this choice will increase the number of patterns that cannot be represented in the aggregated model. □

This aggregation can be done repeatedly. A problem P can be aggregated using $\Delta = \Delta_1$, leading to an aggregated problem P_1^{da} that can in turn be aggregated with $\Delta = \Delta_2$, and so on. Disaggregation is applied to P_i^{da} to recover the set of original adjacent columns of P_{i-1}^{da} or P if $i = 1$.

To solve P, we use a n-phase algorithm that solves the sequence P_{n-1}^{da}, P_{n-2}^{da},..., P by column generation. A limit is imposed in the number of items in each equivalence group. Ideally, this control should be made through the parameters Δ_i. The columns in the final restricted master problem of P_i^{da} are partially disaggregated. For each column that is not in the basis, only one column of the original set of adjacent columns is generated. For those that are in the basis, we generate a subset of original columns keeping the coefficients of the aggregated column. Therefore, if pattern $(0, 0, 0, 2, 1)^T$ in Example 4.5 is in the optimal solution, only $(0, 0, 0, 0, 0, 2, 0, 1, 0)^T$, $(0, 0, 0, 0, 0, 0, 2, 1, 0)^T$, $(0, 0, 0, 0, 0, 2, 0, 0, 1)^T$ and $(0, 0, 0, 0, 0, 0, 2, 0, 1)^T$ will be generated. The resolution of P_{i-1}^{da} starts with this initial set of disaggregated columns. An artificial column with a high cost guarantees the feasibility of the disaggregated master problem. The n^{th} iteration consists in solving the original problem P starting with the disaggregated columns of P_1^{da}.

4.7 Computational Experiments

We report the results of three groups of tests performed on randomly generated instances and other well known instances from the literature. We compare the effectiveness of the different approaches based on the number of column generation iterations and the total computing time.

A starting set of columns was computed through a FFD heuristic, in which rolls were filled in order of decreasing lengths. An artificial column was added to the restricted master problem for the case the heuristic does not provide a valid solution due to the rolls' availability constraints. The knapsack subproblems were solved using the *mt1r* procedure of Martello and Toth [88]. At most one column per stock length was generated in each iteration.

$w_i =$	80																					1	1		
	70																							1	1
	60	1	1															1	1						
	55			1	1	1														1	1				
	43			1							1	1	1			2									
	38	1			1		2		2		1			1	1										
	33		1			1		2		2		1	1	1	1		3								
	20						1	1				1		1				2		2		1		1	
	16								1	1	1		1		1				2		2		1		1

Figure 4.6: Set of feasible patterns (Example 4.5)

$w_i =$	80						1	
	70							1
	60	2/5				1/2		
	55	3/5				1/2		
	43	1/5	3/9	2				
	38	2/5	7/9					
	33	2/5	8/9		3			
	20		4/9			1	1/2	1/2
	16		5/9			1	1/2	1/2

Figure 4.7: Set of patterns after column aggregation (Example 4.5)

$w_i =$	80						1	
	70							1
	57	1				1		
	38	1	2	2				
	18		1			2	1	1

Figure 4.8: Final set of patterns (Example 4.5)

Our computational experiments were conducted on a 700 MHz Pentium III with 128 Mb of RAM under Windows ME operating system. The algorithms were coded using C++ and CPLEX 6.5 [69].

Table 4.5 illustrates the relative performance of the dual inequalities introduced in Section 4.3. A comparison is made among the standard column generation algorithm, column generation with the additional dual inequalities (4.5), and column generation considering inequalities (4.5), (4.6), (4.7) and (4.8). Dual inequalities were added before the first resolution of the LP relaxation with a small perturbation of their right hand side for (4.5) and (4.6), so that the respective columns are at the zero level in the final solution. To limit the number of columns inserted, the maximum cardinality of S has been set to $|S| = 2$. Only one column per item was considered when $|S| = 2$. The tests were carried out on a set of randomly generated problems inspired in the triplets instances from the OR-library [11]. For these instances of the related variable sized bin-packing problem, the solution consists in rolls receiving exactly three items. The only difference between the triplets instances of the OR-library and our test-problems is that a limited number of bins with various capacities are now considered. Six groups of 50 instances each were generated with m ranging from 100 to 180 and K from 5 to 15. The items have integer sizes between 100 and 360, while the roll lengths vary between 400 and 720.

Table 4.5 shows the average computing time in seconds (t_{lp}), and the average number of column generation iterations (sp_{lp}) obtained with the three alternative strategies. A percentage of reduction (% *red*) in time and iterations is indicated, comparing the standard column generation algorithm to the other two methods.

The inequalities on items' dual variables seem to have only a limited impact on those instances. The set of inequalities imposed on the rolls' dual variables appears as a good complement to reach an interesting level of stabilization. For the first group of instances, which is also the harder, with an average number of iterations of 162.6, the combined application of these dual inequalities yields a reduction of approximately 50% in the computing time and in the number of subproblems solved. Note that the instances were generated such that the constraints on rolls' availability were truly effective. Indeed, dual inequalities (4.6), (4.7) and (4.8) will be stronger when these restrictions are tighter. In practice, we will surely found many problems with this characteristic.

In Table 4.6, we report the average computational results obtained for a subset of the instances used in a recent publication of Belov and Scheithauer [13]. These results illustrate the impact of the two-phase algorithm based on the aggregation scheme RA of Section 4.6.1 (procedure RA, for short) compared to the standard column generation algorithm with and without the dual inequalities discussed in this article. We chose 15 groups of 50 instances each, with $m = 100$ and $m = 150$, and a number of stock

m	k	Strategy	sp_{lp}	% *red*	t_{lp}	% *red*
		Standard Column Generation	162.6		16.9	
168.0	5.0	Ineq. on Items' Dual Variables	132.1	18.8	14.7	13.0
		Ineq. on Items and Rolls' Dual Variables	80.0	50.8	8.7	48.5
		Standard Column Generation	50.8		3.2	
102.6	11.0	Ineq. on Items' Dual Variables	49.5	2.6	3.5	-9.4
		Ineq. on Items and Rolls' Dual Variables	35.4	30.3	2.4	25.0
		Standard Column Generation	72.3		7.6	
128.0	11.0	Ineq. on Items' Dual Variables	64.2	11.2	7.4	2.6
		Ineq. on Items and Rolls' Dual Variables	45.5	37.1	4.8	36.8
		Standard Column Generation	85.9		13.2	
152.8	11.0	Ineq. on Items' Dual Variables	78.6	8.5	13.6	-3.0
		Ineq. on Items and Rolls' Dual Variables	54.6	36.4	8.2	37.9
		Standard Column Generation	108.0		25.2	
177.9	11.0	Ineq. on Items' Dual Variables	98.1	9.2	25.4	-0.8
		Ineq. on Items and Rolls' Dual Variables	64.2	40.6	14.8	41.3
		Standard Column Generation	87.0		24.7	
179.5	15.0	Ineq. on Items' Dual Variables	84.1	3.3	26.4	-6.9
		Ineq. on Items and Rolls' Dual Variables	62.7	27.9	18.3	25.9

Table 4.5: Performance of the inequalities on items' and rolls' dual variables

lengths varying between 2 and 10.

With procedure RA, the reduction in the number of column generation iterations achieved with dual inequalities is almost always duplicated. For one group of instances ($m = 99.40$ and $K = 3.80$), the improvement reaches 36.2%. The computing time does not generally decrease in the same proportion. However, there exists an appreciable number of instances for which the RA procedure gives very good results. In Table 4.7, we compile the results obtained with the best 10 instances within each group. This represents 20% of the initial instances. Surprisingly, we are able to cut between 43.3% and 83.5% of the column generation iterations and, even with the disaggregation overhead, the total computing time is reduced by up to 71.4%. In some runs of the column generation algorithm with dual inequalities, we observe an increase of the computing time. This situation is due to a limited number of instances for which the $mt1r$ algorithm finds some difficulties.

In all the experiments conducted, the n-phase algorithm of Section 4.6.2 (we will also refer to it as procedure DA) yields far more regular results than those achieved with procedure RA. To illustrate this conclusion, we present the results obtained with a set of well known bin-packing instances for which procedure RA gave only marginal improvements. Three problem sets are considered. The Hard28 problems consists in

m	k	Strategy	sp_{lp}	% *red*	t_{lp}	% *red*
99.10	3.70	Standard Column Generation	135.3		11.7	
		Dual Inequalities	110.9	18.0	5.4	53.8
		Procedure RA + Dual Ineq.	86.1	36.4	5.1	56.4
99.08	4.00	Standard Column Generation	121.0		8.6	
		Dual Inequalities	100.9	16.6	5.3	38.4
		Procedure RA + Dual Ineq.	82.2	32.1	6.1	29.1
99.08	3.98	Standard Column Generation	102.3		10.4	
		Dual Inequalities	84.3	17.6	4.7	54.8
		Procedure RA + Dual Ineq.	68.6	32.9	5.1	51.0
99.08	3.73	Standard Column Generation	121.3		5.1	
		Dual Inequalities	99.0	18.4	5.3	-3.9
		Procedure RA + Dual Ineq.	78.9	35.0	5.6	-9.8
99.08	4.00	Standard Column Generation	119.8		8.8	
		Dual Inequalities	99.7	16.8	5.3	39.8
		Procedure RA + Dual Ineq.	80.8	32.6	5.5	37.5
99.03	3.87	Standard Column Generation	136.9		7.3	
		Dual Inequalities	106.3	22.4	4.4	39.7
		Procedure RA + Dual Ineq.	80.6	41.1	3.4	53.4
99.10	3.74	Standard Column Generation	128.1		2.4	
		Dual Inequalities	101.4	20.8	2.7	-12.5
		Procedure RA + Dual Ineq.	81.7	36.2	2.2	8.3
99.10	3.74	Standard Column Generation	125.2		3.6	
		Dual Inequalities	100.4	19.8	3.9	-8.3
		Procedure RA + Dual Ineq.	80.6	35.6	3.4	5.6
99.08	3.78	Standard Column Generation	123.1		4.3	
		Dual Inequalities	100.7	18.2	5.5	-27.9
		Procedure RA + Dual Ineq.	79.8	35.2	4.5	-4.7
99.31	3.77	Standard Column Generation	98.8		6.3	
		Dual Inequalities	74.7	24.4	2.4	61.9
		Procedure RA + Dual Ineq.	44.6	54.9	1.8	71.4
99.40	3.80	Standard Column Generation	81.7		1.4	
		Dual Inequalities	60.5	25.9	1.3	7.1
		Procedure RA + Dual Ineq.	31.0	62.1	0.7	50.0
99.23	6.23	Standard Column Generation	104.0		6.8	
		Dual Inequalities	84.0	19.2	7.0	-2.9
		Procedure RA + Dual Ineq.	64.2	38.3	5.6	17.6
148.50	3.74	Standard Column Generation	182.8		17.2	
		Dual Inequalities	144.4	21.0	15.5	9.9
		Procedure RA + Dual Ineq.	110.7	39.4	13.8	19.8
148.47	3.77	Standard Column Generation	220.6		46.0	
		Dual Inequalities	176.0	20.2	20.5	55.4
		Procedure RA + Dual Ineq.	141.4	35.9	18.9	58.9
99.09	8.67	Standard Column Generation	85.0		8.4	
		Dual Inequalities	71.2	16.2	7.8	7.1
		Procedure RA + Dual Ineq.	59.8	29.6	7.7	8.3

Table 4.6: Performance of aggregation scheme RA

m	k	Strategy	sp_{lp}	% *red*	t_{lp}	% *red*
		Standard Column Generation	104.4		2.7	
99.10	3.50	Dual Inequalities	85.3	18.3	3.6	-33.3
		Procedure RA + Dual Ineq.	42.1	59.7	2.0	25.9
		Standard Column Generation	103.0		3.2	
99.00	4.00	Dual Inequalities	83.0	19.4	3.0	6.3
		Procedure RA + Dual Ineq.	51.4	50.1	2.3	28.1
		Standard Column Generation	94.9		4.1	
99.00	4.00	Dual Inequalities	79.0	16.8	3.3	19.5
		Procedure RA + Dual Ineq.	51.2	46.0	2.6	36.6
		Standard Column Generation	94.3		2.5	
99.00	3.60	Dual Inequalities	74.2	21.3	3.3	-32.0
		Procedure RA + Dual Ineq.	41.4	56.1	2.2	12.0
		Standard Column Generation	105.8		3.0	
99.00	4.00	Dual Inequalities	86.0	18.7	2.9	3.3
		Procedure RA + Dual Ineq.	54.2	48.8	2.1	30.0
		Standard Column Generation	91.3		1.7	
99.00	3.80	Dual Inequalities	72.3	20.8	1.8	-5.9
		Procedure RA + Dual Ineq.	38.4	57.9	1.1	35.3
		Standard Column Generation	100.2		1.5	
99.00	3.60	Dual Inequalities	77.6	22.6	2.3	-53.3
		Procedure RA + Dual Ineq.	43.6	56.5	1.3	13.3
		Standard Column Generation	97.1		1.9	
99.00	3.60	Dual Inequalities	75.4	22.3	2.8	-47.4
		Procedure RA + Dual Ineq.	40.6	58.2	1.4	26.3
		Standard Column Generation	91.8		2.2	
98.90	3.60	Dual Inequalities	74.6	18.7	2.9	-31.8
		Procedure RA + Dual Ineq.	39.7	56.8	1.8	18.2
		Standard Column Generation	86.7		1.6	
99.20	3.80	Dual Inequalities	65.7	24.2	1.4	12.5
		Procedure RA + Dual Ineq.	17.7	79.6	0.5	68.8
		Standard Column Generation	86.5		1.4	
99.30	3.80	Dual Inequalities	65.9	23.8	1.3	7.1
		Procedure RA + Dual Ineq.	14.3	83.5	0.4	71.4
		Standard Column Generation	88.1		3.4	
99.10	6.30	Dual Inequalities	70.7	19.8	3.5	-2.9
		Procedure RA + Dual Ineq.	40.0	54.6	2.1	38.2
		Standard Column Generation	147.0		7.7	
148.40	3.80	Dual Inequalities	116.2	21.0	11.2	-45.5
		Procedure RA + Dual Ineq.	64.4	56.2	5.3	31.2
		Standard Column Generation	171.3		9.2	
148.50	3.70	Dual Inequalities	129.3	24.5	8.7	5.4
		Procedure RA + Dual Ineq.	77.9	54.5	8.3	9.8
		Standard Column Generation	79.2		6.0	
99.30	9.00	Dual Inequalities	66.0	16.7	5.8	3.3
		Procedure RA + Dual Ineq.	44.9	43.3	4.6	23.3

Table 4.7: Computational results for the best 10 instances of each group in Table 4.6

28 instances originally generated by J. Schoenfield [103]. As we can see in Table 4.8, the standard column generation algorithm takes on average 532.4 iterations, which is a considerable number. The two remaining problem sets are the t501 and t249 triplet instances from the already mentioned OR-library [11].

With the instances of the Hard28 set, we were able to control the number of items in each equivalence group using only the parameters Δ_i. In fact, considering a 4 iteration process, equivalence groups with more than 5 items were rare. This is an a priori guarantee that the time needed to perform the disaggregation will never be high. With the triplet instances, the situation is different. Item sizes differ only in small amounts and, for example, an aggregation with $\Delta = 1$ leads to a high number of items going to the same equivalence group. Therefore, the maximum number of items per equivalence group was restricted to 5 elements. Since the number of column generation iterations achieved with dual inequalities is reasonably low, we tested a 2 iterations procedure.

In Table 4.8, 4.9 and 4.10, we present the results obtained with standard column generation, column generation using dual inequalities on the items' dual variables and procedure DA. In the latter, dual inequalities were also added to the aggregated master problems. In the next comments, we will denote these procedures by SCG, CG-DI and DA-DI, respectively.

With procedure DA-DI, the number of column generation iterations needed to reach the optimal solution decreases significantly. Compared to CG-DI, the percentage of reduction is approximately 50% for the three sets of instances. The number of columns in the final restricted master problem increases, but these extra columns are generated faster via the disaggregation processes. Indeed, the computational results show that it is more efficient to perform a restricted disaggregation rather than generating columns by solving a knapsack problem, whatever the algorithm we choose. For the three sets of instances, the average computing times decreases with procedure DA-DI. For the Hard28 set, standard column generation needs 532.4 iterations, on average, and runs in 8.9 seconds. With CG-DI, we get the optimal solution 16.9% faster. Procedure DA-DI almost doubles this value with a time reduction of 32.6%. For the t501 set, CG-DI yields a reduction of 32.8%, while procedure DA-DI improves this value in 25.2%. The maximum reduction in the computing time is achieved for the instance t501_02 with 69.2%. For this instance, procedure CG-DI yields a reduction of 25.6%. With problems growing in size and complexity, this tendency in time reduction will surely become even more significant.

4.8 Conclusion

In this chapter, we analyzed different strategies to stabilize and accelerate column generation in the context of multiple length cutting stock problems, and variable sized

Problem	m	Standard CG		Dual Inequalities				procedure DA + Dual Inequalities			
		sp_{lp}	t_{lp}	sp_{lp}	% *red*	t_{lp}	% *red*	sp_{lp}	% *red*	t_{lp}	% *red*
bpp14	136	472	6.1	290	38.6	4.6	24.6	142	69.9	3.2	47.5
bpp832	139	497	6.9	317	36.2	5.4	21.7	172	65.4	5.6	18.8
bpp40	144	594	9.2	369	37.9	10.9	-18.5	199	66.5	8.2	10.9
bpp360	148	419	5.3	207	50.6	3.4	35.8	53	87.4	1.7	67.9
bpp645	141	561	8.4	344	38.7	6.2	26.2	174	69.0	4.3	48.8
bpp742	148	394	4.7	229	41.9	3.9	17.0	87	77.9	3.5	25.5
bpp766	143	521	7.3	363	30.3	6.3	13.7	187	64.1	5.4	26.0
bpp60	144	518	7.1	292	43.6	4.8	32.4	161	68.9	4.5	36.6
bpp13	161	682	12.6	413	39.4	9.5	24.6	260	61.9	9.9	21.4
bpp195	161	596	10.9	446	25.2	13.7	-25.7	282	52.7	9.6	11.9
bpp709	160	550	9.3	379	31.1	10.5	-12.9	182	66.9	6.9	25.8
bpp785	163	679	13.2	459	32.4	10.9	17.4	251	63.0	8.9	32.6
bpp47	158	476	7	220	53.8	3.9	44.3	104	78.2	3.4	51.4
bpp181	157	546	8.4	363	33.5	7.1	15.5	176	67.8	5.9	29.8
bpp359	164	417	5.9	252	39.6	4.9	16.9	142	65.9	4.2	28.8
bpp485	163	552	9	329	40.4	6.8	24.4	166	69.9	5.7	36.7
bpp640	165	396	5.3	211	46.7	4.1	22.6	53	86.6	2.4	54.7
bpp716	158	383	4.8	206	46.2	3.8	20.8	86	77.5	2.9	39.6
bpp119	173	629	12.5	371	41.0	9.2	26.4	274	56.4	10.9	12.8
bpp144	173	548	10.4	354	35.4	9.2	11.5	185	66.2	7.3	29.8
bpp561	177	598	12.4	414	30.8	11.5	7.3	273	54.3	9.9	20.2
bpp781	174	606	12.5	374	38.3	10.5	16.0	167	72.4	6.5	48.0
bpp900	173	632	12.6	403	36.2	10.5	16.7	168	73.4	6.3	50.0
bpp175	185	528	9.3	264	50.0	6.2	33.3	154	70.8	6.6	29.0
bpp178	178	608	11.7	357	41.3	9.1	22.2	248	59.2	10.4	11.1
bpp419	189	673	14.5	390	42.1	11.2	22.8	201	70.1	7.9	45.5
bpp531	175	443	6.7	220	50.3	4.7	29.9	55	87.6	2.4	64.2
bpp814	179	388	5.7	208	46.4	4.3	24.6	68	82.5	3.0	47.4
average	161.8	532.4	8.9	323.0	39.3	7.4	16.9	166.8	68.7	6.0	32.6

Table 4.8: Solution data for the Hard28 instances

Problem	m	Standard CG		Dual Inequalities				procedure DA + Dual Inequalities			
		sp_{lp}	t_{lp}	sp_{lp}	% *red*	t_{lp}	% *red*	sp_{lp}	% *red*	t_{lp}	% *red*
t501_00	190	173	11.3	104	39.9	7.5	33.6	44	74.6	5.3	53.1
t501_01	192	152	9.6	87	42.8	6.6	31.3	38	75.0	4.3	55.2
t501_02	190	174	11.7	100	42.5	8.7	25.6	43	75.3	3.6	69.2
t501_03	199	169	15.7	103	39.1	14.1	10.2	55	67.5	5.7	63.7
t501_04	195	168	11.4	96	42.9	12.4	-8.8	44	73.8	6.0	47.4
t501_05	195	182	12.3	96	47.3	7.4	39.8	54	70.3	4.5	63.4
t501_06	196	180	12.4	96	46.7	7.4	40.3	44	75.6	6.1	50.8
t501_07	192	168	10.8	91	45.8	6.7	38.0	38	77.4	4.4	59.3
t501_08	196	170	11.4	103	39.4	8.0	29.8	47	72.4	4.4	61.4
t501_09	189	155	9.8	96	38.1	7.2	26.5	42	72.9	3.8	61.2
t501_10	190	165	10.6	92	44.2	6.8	35.8	39	76.4	4.7	55.7
t501_11	195	178	12.0	102	42.7	7.9	34.2	44	75.3	6.1	49.2
t501_12	189	172	10.9	91	47.1	6.5	40.4	48	72.1	5.0	54.1
t501_13	198	187	13.1	104	44.4	8.3	36.6	47	74.9	6.3	51.9
t501_14	203	190	14.4	95	50.0	7.7	46.5	45	76.3	4.8	66.7
t501_15	197	181	13.0	102	43.6	8.1	37.7	39	78.5	6.1	53.1
t501_16	198	176	11.9	95	46.0	7.2	39.5	48	72.7	6.0	49.6
t501_17	196	171	12.0	92	46.2	7.7	35.8	43	74.9	3.7	69.2
t501_18	193	189	12.4	99	47.6	7.4	40.3	52	72.5	4.1	66.9
t501_19	192	182	11.6	95	47.8	7.0	39.7	50	72.5	5.7	50.9
average	194.3	174.1	11.9	97.0	44.3	8.0	32.8	45.2	74.0	5.0	58.0

Table 4.9: Solution data for the t501 instances

Problem	m	Standard CG		Dual Inequalities				procedure DA + Dual Inequalities			
		sp_{lp}	t_{lp}	sp_{lp}	% *red*	t_{lp}	% *red*	sp_{lp}	% *red*	t_{lp}	% *red*
t249_00	134	146	4.2	82	43.8	2.5	40.5	48	67.1	2.1	50.0
t249_01	140	151	4.5	87	42.4	3.8	15.6	49	67.5	2.1	53.3
t249_02	139	152	4.5	89	41.5	2.9	35.6	46	69.7	2.2	51.1
t249_03	142	150	4.5	75	50.0	2.5	44.4	48	68.0	2.3	48.9
t249_04	134	136	3.5	76	44.1	2.3	34.3	42	69.1	2.0	42.9
t249_05	145	158	5.0	80	49.4	2.8	44.0	41	74.1	3.0	40.0
t249_06	138	148	4.1	84	43.2	2.7	34.1	47	68.2	1.6	61.0
t249_07	137	136	3.8	80	41.2	2.6	31.6	42	69.1	2.9	23.7
t249_08	139	153	4.5	93	39.2	4.8	-6.7	43	71.9	2.0	55.6
t249_09	141	155	5.0	92	40.7	3.2	36.0	41	73.5	1.8	64.0
t249_10	140	146	4.0	83	43.2	2.7	32.5	49	66.4	2.4	40.0
t249_11	141	149	4.3	83	44.3	2.8	34.9	38	74.5	2.2	48.8
t249_12	141	146	4.2	83	43.2	2.9	31.0	42	71.2	2.0	52.4
t249_13	141	153	4.6	84	45.1	2.8	39.1	42	72.5	2.0	56.5
t249_14	145	157	5.1	103	34.4	3.8	25.5	54	65.6	2.3	54.9
t249_15	142	145	4.3	80	44.8	2.6	39.5	49	66.2	1.9	55.8
t249_16	144	151	4.6	81	46.4	2.8	39.1	43	71.5	2.2	52.2
t249_17	145	158	4.9	92	41.8	3.2	34.7	43	72.8	2.3	53.1
t249_18	138	144	4.1	82	43.1	2.7	34.1	48	66.7	1.6	61.0
t249_19	136	137	3.6	77	43.8	2.6	27.8	41	70.1	1.9	47.2
average	140.1	148.6	4.4	84.3	43.3	3.0	31.8	44.8	69.9	2.1	52.3

Table 4.10: Solution data for the t249 instances

bin-packing problems. Some of the prescribed methods are directly applicable to the standard cutting stock problem (bin-packing problem, respectively), the well known special case where only one stock length is available.

New weak and deep dual-optimal inequalities were introduced. We showed that they are a good complement to the dual inequalities imposed on items' dual variables. Their number is limited, and so we can normally enumerate them all before the resolution of the first master problem.

An important contribution of this chapter is the application of dual inequalities in all the branching nodes of the branch-and-bound tree. Prior results have shown the potential of dual inequalities in stabilizing and accelerating column generation algorithms. However, these results are limited to the LP relaxation part of the cutting stock problem. We have shown how to extend them to all the branching nodes, and we also gave new valid dual inequalities where the dual variables for the branching constraints are taken into account. Computational experiments were conducted indicating a significant reduction in the total computing time and the number of branching nodes.

Additionally, we explored the idea of aggregation to control the progress of the dual variables and thus accelerate the resolution of the master problems. Procedure RA, despite its extreme simplicity, gave very interesting results, but was outperformed by the more sophisticated n-phase algorithm of Section 4.6.2. This latter procedure forces sets of dual variables to be equal during its various stages. The respective approximations are solved by column generation with reduced tailing-off, which leads to an overall process with better convergence properties.

To evaluate the performance of these approaches, extensive computational experiments were carried out on various sets of randomly generated instances and other instances found in the literature. The appreciable improvements achieved allow us to claim the effectiveness of the strategies based on aggregation.

Chapter 5

The Pattern Minimization Problem

After an optimal cutting plan has been devised, one might want to further reduce the operational costs by minimizing the number of setups associated to this plan. A setup operation occurs each time a different cutting pattern begins to be produced. The related optimization problem is known as the Pattern Minimization Problem, and it is particularly hard to solve exactly. In this chapter, we explore different techniques to strengthen the formulation proposed by Vanderbeck [123]. We use the dual feasible functions described in [42] to derive valid superadditive inequalities for the integer knapsack polytope, from different constraints of the model, including surrogate constraints. We also propose a method to improve these cuts, when all the coefficients are known, or alternatively, when one can easily predict them. We developed an integrated branch-and-price-and-cut algorithm, which allows one to derive stronger cuts by combining the branching constraints with other constraints of the model. Besides, it is also important to note that our branching scheme ensures convergence. The computational experiments were conducted using the instances tested in [123], plus an additional set of randomly generated instances.
We also explored the extension of the algorithm to the case in which multiple stock lengths are available. As for the standard case, the cutting plan must not consume more resources than the optimal plan, and, hence, an upper bound is enforced on the total stock length required. Computational results are reported at the end of the chapter.

Keywords: Pattern Minimization Problem, Column Generation, Surrogate Constraints, Dual Feasible Functions, Cutting Planes, Branch-and-Price-and-Cut

5.1 Introduction

Trim loss has been traditionally considered as the primary objective of cutting stock problems, but other costs may also be relevant. A setup cost, for example, is incurred each time we move from a pattern to a different one in a cutting plan. These changes take time. The knives must be repositioned to fit the next pattern. As a consequence, waste may be generated since various trial runs may be needed to reach the right positions. Finding a good pattern sequencing may minimize these costs by limiting the movements of the knives. This latter problem is referred to in the cutting stock literature as the Knife Change Minimization Problem; it is particularly hard to solve. In this chapter, we address the former problem of minimizing the number of setups or Pattern Minimization Problem (PMP).

Pattern minimization can be tackled with three different priorities: as a primary objective (this is quite rare), combined with the objective of trim loss minimization or as a secondary objective. Making it the main objective of the cutting plan may result in solutions with an increased material usage. In the case of an unlimited (or at least non restrictive) supply of raw material, an obvious solution consists in cutting one item per stock roll. This gives an upper bound equal to the number of different items ordered. Solutions generated using a combined trim loss and setups minimization objective seem to better fit the reality of industrial applications. The cutting plans produced with such an objective take clearly into account the trade-off between waste and setups, and only if the cost of the raw material is really insignificant will the number of setups be individually minimized. Setups can also be minimized in a second phase restricting the number of stock rolls to use to its minimum possible value obtained by solving the corresponding cutting stock problem. Alternatively, a higher value may be chosen. The number of patterns in the solution of the cutting stock problem is an upper bound on the number of setups.

The Pattern Minimization Problem is strongly NP-hard even when the corresponding cutting stock problem has a trivial solution [89]. There are many references concerning heuristic approaches. One of the oldest is perhaps from Haessler [61, 62], who developed a sequential heuristic procedure. A list of candidate patterns is first generated. Patterns are then sequentially added to the solution, selecting first those that have smaller waste and higher frequencies. The number of different patterns in the LP solutions is reduced, which makes the rounding of fractional solutions easier.

Other researchers tried to reduce the number of setups starting with a given solution by combining two or more patterns. Allwood and Goulimis [3], Johnston [72, 73] and Diegel [33] developed such algorithms. Foerster and Waescher [44] proposed recently the KOMBI heuristic, which allows many types of combinations.

Chen *et al.* [19] proposed a slight variation of the basic simulated annealing algo-

rithm to solve a nonlinear model with two-sided demand constraints, different sizes of raw material, load balancing between the cutting machines and a combined trim loss and pattern minimization objective. The authors compared their algorithm to the standard simulated annealing using a small size instance (4 machine roll sizes and 4 ordered items), and performing an extensive statistical analysis of the effects of the parameters. They obtained better objective values in less time. Teghem *et al.* [112] used also a simulated annealing algorithm for the problem of printing book covers with fixed and variable costs.

Umetani *et al.* [113] proposed an iterated local search metaheuristic. The neighborhood is defined by switching one of the cutting patterns from the current solution. The authors devised an adaptive pattern generation heuristic and, in fact, restrict this neighborhood to the set of patterns it generates. The new pattern frequencies are computed with a heuristic based on the nonlinear Gauss-Seidel method. Since the number of setups is fixed, these frequencies are computed such that the sum of squares of the deviations from the demands is minimized. The algorithm was compared to the sequential heuristic procedure of Haessler and to the KOMBI heuristic. The computational experiments conducted indicate a comparable performance.

Goulimis [57] solved the cutting stock problem with two-sided demand constraints using branch-and-bound and cutting planes. He focused on a set of small instances for which the complete set of cutting patterns can be generated in reasonable time. Setups were minimized afterwards, combining patterns using an approach inspired on Johnston [72].

Vanderbeck [123] solved the problem for a fixed number of stock sheets, with all the stocks having the same size. He used a model where columns are patterns with an associated multiplicity, and derived a branch-and-price-and-cut algorithm. The model has an exponential number of columns. Each valid cutting pattern is replicated until the maximum multiplicity that ensures that no more items than those demanded are included. Unless the instances are very small, the problem can only be tackled by column generation. The subproblem is originally nonlinear, but it can be solved as a sequence of integer bounded knapsack problems.

Even if it improves the LP bound of the compact assignment formulation, the lower bound of the model of Vanderbeck remains quite weak. In part, this is due to what the author called LP cheating, *i.e.*, columns with a large multiplicity getting preferentially fractional values in the LP solutions. For a set of real-life instances, the author experienced an integrality gap of 33.5%. He then resorted to general cutting planes to strengthen the linear relaxation. Two superadditive functions were used to derive valid inequalities from single rows, namely the demand constraints and the constraint on the maximum number of stock sheets used. A proof is given showing that these cuts can be stronger than the rank 1 Chvátal-Gomory cuts. The integrality

gap is finally reduced to 13.8%. The gap would certainly be even more reduced if facet defining cover inequalities could be used, but the changes induced to the subproblem would make it intractable.

LP based branch-and-bound is used to find integer solutions. Subsets of columns are selected for branching using 7 different rules. Branching begins with the columns that have a higher multiplicity and a fractional total sum. Fixing first these variables leads to a faster improvement of the lower bounds. However, all these rules do not ensure convergence to an integer optimal solution.

Belov [12] considered the combined trim loss and pattern minimization problem. He used an extension of the Gilmore and Gomory model [52] for the standard cutting stock problem, which has more constraints than columns. Since this is a weaker column generation reformulation of the compact assignment formulation, the author strengthened it using a bound on the pattern frequencies similar to that of Vanderbeck for the pattern multiplicities. He showed that the resulting LP bound is equal to the one given by the formulation finally used by Vanderbeck. To make his model computationally more tractable, Belov removed all the binary variables and corresponding definition constraints and introduced a nonlinear objective function that accounts for the material costs. The objective function is linearly relaxed giving a model with a weak LP bound. The proposed algorithm failed to solve 8 of the 16 instances tested by Vanderbeck, while Vanderbeck was unable to close the integrality gap for only 4 of them.

In this chapter, we analyze an exact branch-and-price-and-cut algorithm for the latter version of PMP. In Section 5.2, the different formulations for the problem are reviewed. In Section 5.2.4, we will show how the largest pattern multiplicity can be reduced in the model of Vanderbeck, and the impact it has on the respective LP bound. We proceed by describing how one of these models, the one due to Vanderbeck [123], can be strengthened, and how to derive cutting planes from the constraints of the model using for the first time the dual feasible functions discussed in [42]. Some enhancements to our cutting plane procedure are finally described. They rely on the principle of surrogate constraints. Additionally, we extended our algorithm to cope with multiple stock lengths.

5.2 Integer Linear Programming Formulations

In this section, we present three possible IP formulations for the PMP. An instance of the PMP consists in a set of items with sizes w_i and a demand of b_i units, $i = 1, \ldots, m$, and rolls of length W. Here, we consider the item sizes ordered by decreasing sizes, from w_1, the largest, to w_m, the smallest. The optimal solution of the corresponding cutting stock problem is denoted by z_{CSP}. We consider this value as an upper bound

on the number of rolls over which the cutting patterns can be distributed.

5.2.1 A Compact Assignment Formulation

The PMP can be formulated as an integer assignment problem with nonlinear constraints as follows

$$\min \sum_{k=1}^{z_{CSP}} y_k \tag{5.1}$$

subject to

$$\sum_{k=1}^{z_{CSP}} z_k x_{ik} = b_i,\ i = 1, \ldots, m, \tag{5.2}$$

$$\sum_{k=1}^{z_{CSP}} z_k \leq z_{CSP}, \tag{5.3}$$

$$z_k \leq z_{CSP} y_k,\ k = 1, \ldots, z_{CSP}, \tag{5.4}$$

$$\sum_{i=1}^{m} w_i x_{ik} \leq W y_k,\ k = 1, \ldots, z_{CSP}, \tag{5.5}$$

$$x_{ik} \geq 0 \text{ and integer},\ i = 1, \ldots, m,\ k = 1, \ldots, z_{CSP}, \tag{5.6}$$

$$y_k \in \{0, 1\},\ k = 1, \ldots, z_{CSP}, \tag{5.7}$$

$$z_k \geq 0 \text{ and integer},\ k = 1, \ldots, z_{CSP}. \tag{5.8}$$

This model was proposed by Vanderbeck in [123]. The nonlinearities are in the demand constraints (5.2). Variables x_{ik} denote the number of items i in pattern k, while z_k is the number of times pattern k is used. Variables y_k are binary and take the value 1 if pattern k is used, and 0 otherwise. Since the objective is to minimize the sum of these y_k variables and z_k are general integer variables, two patterns k_1 and k_2 with $k_1 \neq k_2$ belonging to an optimal solution will be necessarily different. The number of patterns is limited to z_{CSP} units through constraint (5.3). Constraints (5.5) are the knapsack constraints. This model is compact; it has exactly $2z_{CSP} + mz_{CSP}$ variables. In the following sections, we review two possible column generation decompositions for it.

5.2.2 A Gilmore and Gomory based Model

The first column generation formulation is largely inspired on the model from Gilmore and Gomory for the cutting stock problem [52]. A column in this reformulation remains an exact cutting pattern, and is now associated to a binary variable in order to keep track of the exact number of different patterns used. Formally, this model is obtained

by keeping in the master problem constraints (5.2)-(5.4) as follows

$$\min \sum_{p \in P} \mu_p \tag{5.9}$$

subject to

$$\sum_{p \in P} a_{ip} \lambda_p = b_i, \ i = 1, \ldots, m, \tag{5.10}$$

$$\sum_{p \in P} \lambda_p \leq z_{CSP}, \tag{5.11}$$

$$\lambda_p \leq \mu_p \min_{i=1,\ldots,m} \left\lfloor \frac{b_i}{a_{ip}} \right\rfloor, \ p \in P, \tag{5.12}$$

$$\lambda_p \geq 0 \text{ and integer}, \ p \in P, \tag{5.13}$$

$$\mu_p \in \{0, 1\}, \ p \in P. \tag{5.14}$$

being P the pattern set. Coefficients a_{ip} indicate the number of items i in pattern p, variables λ_p denote the usage of pattern p, and μ_p are 1 if pattern p is used, and 0 otherwise. Constraints (5.5) go to the pricing subproblem, which remains a knapsack problem.

Model (5.9)-(5.14) has an exponential number of columns and constraints. In [12], Belov solved a relaxation where constraints (5.12) are dropped, but did not really succeed in solving many of the instances tested in [123]. In [123], Vanderbeck described a weaker version of (5.9)-(5.14), where the coefficient in the right hand side of (5.12) is replaced by a larger upper bound on the number of rolls. He also proposed a computationally more tractable decomposition, which, according to a result from Belov [12], provides an LP bound equivalent to the one obtained with (5.9)-(5.14).

5.2.3 Column Generation Reformulation from Vanderbeck

In this chapter, we consider an alternative column generation model obtained by keeping in the master only the constraints (5.2) and (5.3). This decomposition yields a nonlinear pricing subproblem, which can be linearized by fixing one of its variables. Each column of the master is now a pattern replicated as many times as given by the column multiplicity. The master problem states as follows

$$\min \sum_{p \in P} \sum_{n=1}^{ub(P_p)} \lambda_{pn} \tag{5.15}$$

subject to

$$\sum_{p \in P} \sum_{n=1}^{ub(P_p)} n a_{ip} \lambda_{pn} = b_i, \ i = 1, \ldots, m, \tag{5.16}$$

$$\sum_{p \in P} \sum_{n=1}^{ub(P_p)} n \lambda_{pn} \leq z_{CSP}, \tag{5.17}$$

$$\lambda_{pn} \in \{0, 1\}, \ p \in P, \ n = 1, \ldots, ub(P_p). \tag{5.18}$$

The λ_{pn} variables are double indexed, with p indicating the original pattern in P, and n the number of replications of this pattern. Value n is also denoted as the multiplicity of the column. The maximum admissible multiplicity for a pattern p in P is given by

$$ub(P_p) = \min_{i=1,\ldots,m} \left\lfloor \frac{b_i}{a_{ip}} \right\rfloor .$$

The pricing subproblem is the following nonlinear knapsack problem

$$\max\ n \left(\sum_{i=1}^{m} \pi_i x_i + \rho \right) \tag{5.19}$$

subject to

$$\sum_{i=1}^{m} w_i x_i \leq W, \tag{5.20}$$

$$n x_i \leq b_i, \tag{5.21}$$

$$n \in \{1, \ldots, n^{max}\}, \tag{5.22}$$

$$x_i \geq 0 \text{ and integer}, \tag{5.23}$$

being n^{max} a global upper bound on the patterns multiplicities, and π and ρ the vector of dual variables for the demand constraints (5.16) and the constraint on the number of rolls (5.17), respectively. Vanderbeck suggests the following value for n^{max}

$$n^{max} = \min \left\{ z_{CSP} - \underline{z} + 1, \max_i\ b_i \right\},$$

where $\underline{z}$ is a given lower bound on the minimum number of setups. The nonlinearities of (5.19)-(5.23) can be easily avoided by considering a sequence of bounded integer knapsack problems with a fixed multiplicity n

$$\max\ \sum_{i=1}^{m} \pi_i x_i \tag{5.24}$$

subject to

$$\sum_{i=1}^{m} w_i x_i \leq W, \tag{5.25}$$

$$x_i \leq \min \left\{ \left\lfloor \frac{W}{w_i} \right\rfloor, \left\lfloor \frac{b_i}{n} \right\rfloor \right\},\ i = 1, \ldots, m, \tag{5.26}$$

$$x_i \geq 0 \text{ and integer}. \tag{5.27}$$

Vanderbeck already noticed that, sometimes, it is not necessary to enumerate all the possible values of n, since an optimal solution to (5.24)-(5.27) may remain optimal for successive values of n. However, depending on the extra constraints that may be enforced in the master (branching constraints or cutting planes), a complete enumeration may be unavoidable.

5.2.4 Improving the Model of Vanderbeck

When an upper bound is settled on the number of rolls, as it is the case, we can easily derive an upper bound on the total waste that can be produced by a cutting plan. Following this idea, the model of Vanderbeck can be improved by excluding from the set of admissible patterns those with an excessively high value of waste. Here, excessive means greater than

$$l = z_{CSP}W - \sum_{i=1}^{m} w_i b_i, \tag{5.28}$$

which is precisely the amount of waste generated by a cutting plan with z_{CSP} rolls. Indeed, these patterns will never be part of any optimal integer solution to the corresponding PMP. However, unless this is explicitly enforced, the corresponding columns can be generated and appear with a fractional value in the optimal solution of the LP relaxations. The following new constraint is added to the pricing subproblem to avoid generating these invalid cutting patterns

$$n\left(W - \sum_{i=1}^{m} w_i x_i\right) \leq l. \tag{5.29}$$

Restricting the set of valid columns strengthens the model (5.15)-(5.18), but as computational results will show, the main value of the loss constraint (5.29) seems to be elsewhere. In fact, this constraint may help in reducing the value of n^{max}, and expectably, the number of knapsack problems that have to be solved to price out the attractive columns.

If a model with constraints (5.24)-(5.27) and (5.29) is infeasible for $n = n_1$, this can only be due to (5.29), since at least the null solution is always valid for (5.24)-(5.27). This means that there is no arrangement of items such that the unused space in the knapsack is between 0 and $W - \left\lfloor \frac{l}{n_1} \right\rfloor$. Clearly, the knapsack problems related to multiplicities greater than n_1 are also infeasible. Increasing n reduces the upper bounds on the item frequencies and the interval on which the pattern waste must be. If a solution exists for a multiplicity $n_2 > n_1$, this solution will be feasible for the problem with multiplicity n_1. These conclusions lead to the following reformulation of n^{max}

$$n^{max} = \min\left\{z_{CSP} - \underline{z} + 1, \max_i\ b_i, n'\right\}, \tag{5.30}$$

where n' is the solution of the following problem

$$n' = \max\ n \tag{5.31}$$

subject to

$$\sum_{i=1}^{m} w_i x_i \geq W - \left\lfloor \frac{l}{n} \right\rfloor, \tag{5.32}$$

$$\sum_{i=1}^{m} w_i x_i \leq W, \tag{5.33}$$

$$x_i \leq \min\left\{\left\lfloor\frac{W}{w_i}\right\rfloor, \left\lfloor\frac{b_i}{n}\right\rfloor\right\}, \ i = 1, \ldots, m, \tag{5.34}$$

$$n \geq 0 \text{ and integer}, \tag{5.35}$$

$$x_i \geq 0 \text{ and integer}, \ i = 1, \ldots, m. \tag{5.36}$$

The maximum multiplicity of a cutting pattern, denoted above by $ub(P_p)$, can be restated as

$$ub(P_p) = \min\left\{\min_{i=1,\ldots,m}\left\lfloor\frac{b_i}{a_{ip}}\right\rfloor, \left\lfloor\frac{l}{W - \sum_{i=1}^{m} w_i a_{ip}}\right\rfloor\right\}.$$

To evaluate the impact of the new constraint (5.29), we solved the LP relaxation of (5.15)-(5.18) for the instances in [123], with and without this constraint. Table 5.1 summarizes the results. Column $Waste$ lists, for each instance, the waste generated by the optimal solution of the corresponding cutting stock problem. Column kp_{LP} reports the total number of knapsack problems solved. The number of generated columns ($cols_{LP}$), the maximum multiplicity among the added columns ($mult$) and the optimal LP solution (z_{LP}) are also reported. For these instances, the LP bound improves only 0.4%, on average, a value that is quite insignificant. On the other hand, the number of column generation subproblems solved is significantly reduced. While 880 knapsack problems are needed on average to reach the optimal LP solution, with constraint (5.29) this value goes down to 585, a saving of almost 34%. Furthermore, there are 13% less columns generated and the maximum multiplicity of the newly generated columns decreases almost 30%.

5.3 New General Cutting Planes

Clearly, constraint (5.29) is not enough to improve the LP bound given by the linear relaxation of (5.15)-(5.18) satisfactorily. A poor lower bound compromises the chances of a branch-and-bound algorithm. In this section, we explore a new family of valid inequalities used to strengthen model (5.15)-(5.18). Based on the theory of superadditive functions, we prove their validity, and show than they can outperform the cuts of Vanderbeck [123]. Before describing these inequalities, we briefly recall the underlying theory of superadditive functions.

5.3.1 Superadditive functions

In [93], Nemhauser and Wolsey proved that valid inequalities for $\mathbb{Z}^n \cap \{x \in \mathbb{R}^n_+ : Ax \leq b\}$ can be derived applying a function $F : D \subseteq R^m \rightarrow R^1$ over all the coefficients of a

		Without constraint (5.29)				With constraint (5.29)			
Name	*Waste*	kp_{LP}	$cols_{LP}$	*mult*	z_{LP}	kp_{LP}	$cols_{LP}$	*mult*	z_{LP}
kT03	3088	110	30	31	4.77	99	26	31	4.77
kT05	102108	120	42	25	5.65	114	42	15	5.65
kT01	274	160	34	13	2.00	95	22	10	2.10
kT02	16575	468	114	13	15.93	461	112	13	15.93
kT04	36703	295	86	9	6.71	253	100	8	6.74
d16p6	36693	295	86	9	6.71	253	100	8	6.74
7p18	3826	264	44	84	3.74	155	32	56	3.74
d33p20	39905	869	160	13	6.05	583	136	8	6.18
12p19	526	643	98	13	2.88	489	90	12	2.89
d43p21	39764	1204	210	17	7.86	1018	202	13	7.86
kT06	1981	833	76	37	1.72	472	54	32	1.75
kT07	4990	1061	104	55	2.86	640	78	36	2.86
14p12	190500	1230	118	50	3.72	766	98	29	3.75
kT09	699	1577	130	94	3.65	1203	118	64	3.65
11p4	19000	1775	124	85	2.47	806	72	57	2.48
30p0	2562	3176	238	61	5.50	1946	194	36	5.51
avg.	31199.63	880.00	105.88	38.06	5.14	584.56	92.25	26.75	5.16

Table 5.1: Measuring the impact of constraint (5.29)

single constraint. For this purpose, F must be superadditive and nondecreasing. Superadditivity is defined over set D as follows. If F is such that

$$F(x) + F(y) \leq F(x+y), \qquad \forall x, y, x+y \in D,$$

then F is a superadditive function. On the other hand, F is nondecreasing over D if

$$x \leq y \Rightarrow F(x) \leq F(y), \qquad \forall x, y \in D.$$

It follows that any nondecreasing linear function can generate valid inequalities.

Superadditive functions can also be obtained by the composition of other superadditive functions. The inequalities generated by a superadditive function are called superadditive valid inequalities. The maximal inequalities for $\mathbb{Z}^n \cap \left\{x \in \mathbb{R}^n_+ : Ax \leq b\right\}$ are necessarily superadditive valid inequalities.

In [123], Vanderbeck gives a set of valid superadditive inequalities for $S = \{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b,\ a \in \mathbb{N}^n,\ b \in \mathbb{N},\ a_i \leq b, \forall i\}$. Using the superadditive function $F^\gamma(z) = \max\left\{0, \left\lceil \frac{\gamma z}{b} \right\rceil - 1\right\}$, with $\gamma \in \{2, \ldots, b\}$, he derives the inequalities

$$\sum_i \left(\left\lceil \frac{\gamma a_i}{b} \right\rceil - 1 \right) x_i \leq \gamma - 1, \tag{5.37}$$

and uses them to strengthen the linear relaxation of (5.15)-(5.18). In the following section, we describe different sets of superadditive functions that can yield stronger inequalities for S.

5.3.2 A Class of Valid Inequalities for the Integer Knapsack Polytope

The term "dual feasible function" was originally used by Lueker [84]. It denotes a function $f : [0,1] \to [0,1]$, such that for $S \subseteq [0,1]$, the following holds

$$\sum_{x \in S} x \leq 1 \Rightarrow \sum_{x \in S} f(x) \leq 1.$$

Quite recently, Fekete and Schepers [42] used different dual feasible functions to derive fast lower bounds for bin-packing problems. We will show that these functions are also superadditive and nondecreasing, and can consequently be used to generate valid inequalities for integer knapsack polytopes. To the best of our knowledge, dual feasible functions have never been used for this purpose.

All the dual feasible functions described in [42] are based on rounding. The first one slightly improves a function proposed earlier by Lueker. Denoting it by $u^{(k)}$, we can state it as follows

$$\begin{aligned} u^{(k)} : \ & [0,1] \to [0,1] \\ & x \mapsto \begin{cases} x, & \text{for } (k+1)x \in \mathbb{Z}, \\ \frac{\lfloor (k+1)x \rfloor}{k}, & \text{otherwise,} \end{cases} \end{aligned}$$

with $k \in \mathbb{N}$. Function $u^{(k)}$ is clearly nondecreasing. Its superadditivity is proved next.

Proposition 5.1 *Function $u^{(k)}$ is superadditive over $[0,1]$, for $k \in \mathbb{N}$.*

Proof: Let x, y, and $x+y$ be nonnegative real values belonging to $\mathbb{R}_+ \cap [0,1]$. For $(k+1)(x+y) \notin \mathbb{Z}$, we have

$$\begin{aligned} & u^{(k)}(x+y) = \frac{\lfloor (k+1)(x+y) \rfloor}{k} = \\ & = \begin{cases} \frac{\lfloor (k+1)x \rfloor}{k} + \frac{\lfloor (k+1)y \rfloor}{k}, & \text{if } (k+1)(x+y) - (\lfloor (k+1)x \rfloor + \lfloor (k+1)y \rfloor) < 1, \\ \frac{\lfloor (k+1)x \rfloor}{k} + \frac{\lfloor (k+1)y \rfloor}{k} + \frac{1}{k}, & \text{if } (k+1)(x+y) - (\lfloor (k+1)x \rfloor + \lfloor (k+1)y \rfloor) \geq 1. \end{cases} \end{aligned}$$

Hence, the following cases may arise:

1. $(k+1)x \in \mathbb{Z}$ and $(k+1)y \notin \mathbb{Z}$: since $(k+1)(x+y) \notin \mathbb{Z}$ and $\frac{\lfloor (k+1)x \rfloor}{k} = \frac{(k+1)x}{k} > x$, we have

$$u^{(k)}(x) + u^{(k)}(y) = x + \frac{\lfloor (k+1)y \rfloor}{k} < u^{(k)}(x+y);$$

2. $(k+1)x \notin \mathbb{Z}$, $(k+1)y \notin \mathbb{Z}$ and $(k+1)(x+y) \notin \mathbb{Z}$: we have

$$u^{(k)}(x) + u^{(k)}(y) = \frac{\lfloor (k+1)x \rfloor}{k} + \frac{\lfloor (k+1)y \rfloor}{k} \leq u^{(k)}(x+y);$$

Additionally, we may consider the following cases:

1. $(k+1)x \in \mathbb{Z}$ and $(k+1)y \in \mathbb{Z}$: this implies $(k+1)(x+y) \in \mathbb{Z}$, and

$$u^{(k)}(x) + u^{(k)}(y) = x + y = u^{(k)}(x+y);$$

2. $(k+1)x \notin \mathbb{Z}$, $(k+1)y \notin \mathbb{Z}$ and $(k+1)(x+y) \in \mathbb{Z}$: this implies

$$(k+1)(x+y) - (\lfloor (k+1)x \rfloor + \lfloor (k+1)y \rfloor) = 1,$$

Rearranging the terms, we get

$$x + y = \frac{\lfloor (k+1)x \rfloor}{k} + \frac{\lfloor (k+1)y \rfloor}{k} + \frac{1 - (x+y)}{k}.$$

Since $x + y$ must be in $\mathbb{R}_+ \cap [0,1]$, we have finally

$$u^{(k)}(x) + u^{(k)}(y) = \frac{\lfloor (k+1)x \rfloor}{k} + \frac{\lfloor (k+1)y \rfloor}{k} \leq x + y = u^{(k)}(x+y).$$

□

With the next proposition, we show that $u^{(k)}$ generates valid superadditive inequalities that are at least as strong as (5.37).

Proposition 5.2 *For* $S = \{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b,\ a \in \mathbb{N}^n,\ b \in \mathbb{N},\ a_i \leq b,\ \forall i\}$ *and* $k \in \mathbb{N}$*, the following inequality*

$$\sum_i u^{(k)}\left(\frac{a_i}{b}\right) x_i \leq 1 \tag{5.38}$$

is equivalent or dominates (5.37).

Proof: Let $z_i = \frac{a_i}{b}$ and, without loss of generality, assume that $k = \gamma - 1$. Inequalities (5.37) can be rewritten as follows

$$\sum_i \frac{\lceil \gamma z_i \rceil - 1}{\gamma - 1}\, x_i \leq 1.$$

For $\gamma z_i \notin \mathbb{Z}$, we have $\frac{\lceil \gamma z_i \rceil - 1}{\gamma - 1} = \frac{\lfloor \gamma z_i \rfloor}{\gamma - 1} = \frac{\lfloor (k+1) z_i \rfloor}{k} = u^{(k)}(z_i)$. On the other hand, for $\gamma z_i \in \mathbb{Z}$, we have $u^{(k)}(z_i) = z_i \geq \frac{\lceil \gamma z_i \rceil - 1}{\gamma - 1}$, since $z_i \leq 1$ and

$$\frac{\lceil \gamma z_i \rceil - 1}{\gamma - 1} = \frac{\lceil (k+1) z_i \rceil - 1}{k} = \frac{(k+1) z_i - 1}{k} = z_i + \frac{z_i}{k} - \frac{1}{k}.$$

□

Another dual feasible function discussed in [42] is the one that formalizes the procedure developed by Martello and Toth [88] to derive the well known L2 lower bound for bin-packing. This function is also superadditive and nondecreasing, but the inequalities it generates for $S = \{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b,\ a \in \mathbb{N}^n,\ b \in \mathbb{N},\ a_i \leq b,\ \forall i\}$ can now be weaker or stronger than (5.37), depending on the coefficients a_i and b of the original knapsack constraint. We denote this function by $u_1^{(\epsilon)}$; its definition follows

$$u_1^{(\epsilon)}: \quad [0,1] \to [0,1]$$
$$x \mapsto \begin{cases} 0, & \text{for } x < \epsilon, \\ x, & \text{for } \epsilon \leq x \leq 1-\epsilon, \\ 1, & \text{for } x > 1-\epsilon, \end{cases}$$

with $\epsilon \in \left[0, \frac{1}{2}\right]$.

Proposition 5.3 *Function $u_1^{(\epsilon)}$ is superadditive over $[0,1]$, for $\epsilon \in [0, \frac{1}{2}]$.*

Proof: Let x, y, $x+y \in \mathbb{R}_+ \cap [0,1]$ and consider the following three cases:

1. $x + y < \epsilon$: in this case, we have $u_1^{(\epsilon)}(x+y) = 0$ and, since $x < \epsilon$ and $y < \epsilon$, the following holds
$$u_1^{(\epsilon)}(x) + u_1^{(\epsilon)}(y) = 0 = u_1^{(\epsilon)}(x+y);$$

2. $\epsilon \leq x + y \leq 1 - \epsilon$: then $x \leq 1-\epsilon$, $y \leq 1-\epsilon$, and consequently
$$u_1^{(\epsilon)}(x) + u_1^{(\epsilon)}(y) \leq x + y = u_1^{(\epsilon)}(x+y);$$

3. $x + y > 1 - \epsilon$: since $\epsilon \leq \frac{1}{2}$ and $x + y \in [0,1]$, if $x > 1-\epsilon$, then $y < \epsilon$ and
$$u_1^{(\epsilon)}(x) + u_1^{(\epsilon)}(y) = 1 = u_1^{(\epsilon)}(x+y),$$
otherwise
$$u_1^{(\epsilon)}(x) + u_1^{(\epsilon)}(y) \leq x + y \leq 1 = u_1^{(\epsilon)}(x+y).$$

□

The following example shows through two different simple cases that, sometimes, inequalities (5.37) can be dominated by the following inequalities generated with $u_1^{(\epsilon)}$ for $S = \{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b,\ a \in \mathbb{N}^n,\ b \in \mathbb{N},\ a_i \leq b,\ \forall i\}$

$$\sum_i u_1^{(\epsilon)}\left(\frac{a_i}{b}\right) x_i \leq 1. \tag{5.39}$$

We also show that the converse may be true.

Example 5.1 For $S = \{x \in \mathbb{N}^2 : 50x_1 + 52x_2 \leq 100\}$ and $\epsilon = 0.5$, we obtain the valid superadditive inequality $0.5x_1 + x_2 \leq 1$ applying $u_1^{(\epsilon)}$ on the single knapsack constraint of S ($u_1^{(\epsilon)}$ is applied on the coefficients normalized to 1, *i.e.*, divided by the right-hand side of the original inequality). It is easy to confirm that there is no $\gamma \in [2, 100]$ for (5.37) leading to a stronger inequality. Consider now $S = \{x \in \mathbb{N}^2 : 10x_1+19x_2 \leq 100\}$. With $\gamma = 11$, we have an inequality of type (5.37) with the form $0.1x_1 + 0.2x_2 \leq 1$, while with $u_1^{(\epsilon)}$ we are unable to generate any cut stronger than $0.1x_1 + 0.19x_2 \leq 1$, *i.e.*, the original inequality. □

The last dual feasible function is defined with a parameter ϵ in the range $[0, \frac{1}{2}($, and is denoted by $u_2^{(\epsilon)}$.

$$u_2^{(\epsilon)} : \quad [0,1] \to [0,1]$$
$$x \mapsto \begin{cases} 0, & \text{for } x < \epsilon, \\ \frac{1}{\lfloor \epsilon^{-1} \rfloor}, & \text{for } \epsilon \leq x < \frac{1}{2}, \\ \frac{1}{2}, & \text{for } x = \frac{1}{2}, \\ 1 - \frac{\lfloor (1-x)\epsilon^{-1} \rfloor}{\lfloor \epsilon^{-1} \rfloor}, & \text{for } x > \frac{1}{2}. \end{cases}$$

Using a small example, we show that $u_2^{(\epsilon)}$ is not superadditive for all the values of ϵ in $[0, \frac{1}{2}($. However, we can prove that it is superadditive for a smaller interval for ϵ.

Example 5.2 Let ϵ in $u_2^{(\epsilon)}$ be equal to 0.1. For $x = 0.15$ and $y = 0.15$, we have

$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) = 0.2 > u_2^{(\epsilon)}(x + y) = 0.1,$$

which clearly violates the superadditivity condition. □

Proposition 5.4 *Function $u_2^{(\epsilon)}$ is superadditive over $[0, 1]$, for $\epsilon \in]\frac{1}{4}, \frac{1}{2}($.*

Proof: Consider the following four cases:

1. $x + y < \epsilon$: then $x < \epsilon$, $y < \epsilon$ and
$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) = 0 = u_2^{(\epsilon)}(x + y);$$

2. $\epsilon \leq x + y < \frac{1}{2}$: then $x < \frac{1}{2}$ and $y < \frac{1}{4}$ (or vice versa). We have $u_2^{(\epsilon)}(x) \leq \frac{1}{\lfloor \epsilon^{-1} \rfloor}$, $u_2^{(\epsilon)}(y) = 0$, since $\epsilon > \frac{1}{4}$, and hence
$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) \leq \frac{1}{\lfloor \epsilon^{-1} \rfloor} = u_2^{(\epsilon)}(x + y);$$

3. $x + y = \frac{1}{2}$: if $x = \frac{1}{2}$, then $y = 0$ (or vice versa), and

$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) = \frac{1}{2} = u_2^{(\epsilon)}(x + y).$$

On the other hand, if $x < \frac{1}{2}$, then $y < \frac{1}{4}$, (or vice versa); $u_2^{(\epsilon)}(x) \leq \frac{1}{\lfloor \epsilon^{-1} \rfloor} \leq \frac{1}{2}$, $u_2^{(\epsilon)}(y) = 0$, and hence

$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) \leq \frac{1}{2} = u_2^{(\epsilon)}(x + y);$$

4. $x + y > \frac{1}{2}$: with $\epsilon \in]\frac{1}{4}, \frac{1}{2}[$, we have

$$\frac{2}{\lfloor \epsilon^{-1} \rfloor} \leq 1 - \frac{\lfloor (1 - (x + y))\epsilon^{-1} \rfloor}{\lfloor \epsilon^{-1} \rfloor} = u_2^{(\epsilon)}(x + y) \leq 1,$$

and hence, superadditivity is proven for the cases where x and y are both strictly less than $\frac{1}{2}$. If x and y are both equal to 1, we have

$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) = 1 = u_2^{(\epsilon)}(x + y)$$

For $x > \frac{1}{2}$ and $y < \epsilon$, we have

$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) = 1 - \frac{\lfloor (1 - x)\epsilon^{-1} \rfloor}{\lfloor \epsilon^{-1} \rfloor} \leq 1 - \frac{\lfloor (1 - (x + y))\epsilon^{-1} \rfloor}{\lfloor \epsilon^{-1} \rfloor} = u_2^{(\epsilon)}(x + y),$$

The remaining case is when $x > \frac{1}{2}$ and $\epsilon \leq y < \frac{1}{2}$. We have

$$\lfloor (1 - (x + y))\epsilon^{-1} \rfloor = \lfloor (1 - x)\epsilon^{-1} - y\epsilon^{-1} \rfloor \leq \lfloor (1 - x)\epsilon^{-1} - 1 \rfloor = \lfloor (1 - x)\epsilon^{-1} \rfloor - 1,$$

and hence

$$u_2^{(\epsilon)}(x) + u_2^{(\epsilon)}(y) = 1 - \frac{\lfloor (1 - x)\epsilon^{-1} \rfloor}{\lfloor \epsilon^{-1} \rfloor} + \frac{1}{\lfloor \epsilon^{-1} \rfloor} \leq 1 - \frac{\lfloor (1 - (x + y))\epsilon^{-1} \rfloor}{\lfloor \epsilon^{-1} \rfloor}.$$

□

Once more, $u_2^{(\epsilon)}$ is clearly nondecreasing and, for $\epsilon \in]\frac{1}{4}, \frac{1}{2}[$, the following inequalities it generates for $S = \{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b,\ a \in \mathbb{N}^n,\ b \in \mathbb{N},\ a_i \leq b,\ \forall i\}$

$$\sum_i u_2^{(\epsilon)}\left(\frac{a_i}{b}\right) x_i \leq 1, \tag{5.40}$$

can dominate or be dominated by inequalities (5.37), as is shown in the following example.

Example 5.3 For a set $S = \{x \in \mathbb{N}^2 : 10x_1 + 19x_2 \leq 100\}$, function $u_2^{(\epsilon)}$ with $\epsilon \in]\frac{1}{4}, \frac{1}{2}[$ produces no useful valid inequality. However, for the set $S = \{x \in \mathbb{N}^2 : 270x_1 + 745x_2 \leq 1000\}$ and $\epsilon = 0.26$, $u_2^{(\epsilon)}$ yields the inequality $\frac{1}{3}x_1 + x_2 \leq 1$, which is stronger than any inequality of type (5.37). □

5.3.3 Separation Procedures

Let $S = \{x \in \mathbb{N}^n : \sum_i a_i x_i \leq b,\ a \in \mathbb{N}^n,\ b \in \mathbb{N},\ a_i \leq b, \forall i\}$, and x^* be a point in the convex hull of S. To find an inequality of type (5.38) violated by x^*, we simply search sequentially for a value of k from 1 up to $3b$, and check the validity of x^* for the corresponding inequality. The separation procedures for (5.39) and (5.40) are a little more sophisticated.

For (5.39), we adopted a separation procedure inspired on [88]. Let w be the set composed by the elements of $a = (a_1, a_2, \ldots, a_n)$ in decreasing order, *i.e.*, $w_i \geq w_{i+1}$, for $i = 1, \ldots, n-1$, and let $i^* = \min\left\{i \in [1, \ldots, n] : w_i \leq \frac{1}{2}\right\}$. Violated inequalities of type (5.39) are searched sequentially starting with $\epsilon = w_{i^*}$ and ending with $\epsilon = w_n$.

We search for violated inequalities of type (5.40) by selecting ϵ in $u_2^{(\epsilon)}$ from a set S' composed by the following values:

. 0.5-ϱ ;

. a_i, for $a_i \in a = (a_1, a_2, \ldots, a_n)$ and $a_i < \frac{1}{2}$;

. $\frac{1}{p}$, for $p \in \mathbb{N}$;

. $\frac{1-a_i}{p}$, for $a_i \in a = (a_1, a_2, \ldots, a_n)$, $a_i > \frac{1}{2}$ and $p \in \mathbb{N}$.

The values of S' are selected in decreasing order. Coefficient p is a parameter and ρ is a small positive value. In our implementation, we restrict p to 30 units.

5.3.4 Improving the Dual Feasible Functions

In the context of bin-packing problems, the essential property of dual feasible functions can be stated as follows: applying a dual feasible function to the item sizes of any feasible packing solution, the resulting item sizes must continue to fit together in the same bin. These functions are based on rounding: while some of the item sizes are increased, others are decreased accordingly. Given a set $S \subseteq [0, 1]$ of item sizes to pack on a bin of normalized capacity 1, if all the elements of S are known a priori, it may be possible to obtain a better set of transformed sizes, depending on the elements of S. We show how this can be done using an example.

Example 5.4 Consider the function $u^{(k)}$ with $k = 5$ represented in Figure 5.1. A loss zone [42] is a subinterval of $[0, 1]$ for which $u^{(k)}(x) < x$. Win zones correspond to those intervals where $u^{(k)}(x) > x$. In Figure 5.1, loss and win zones are the areas in dark and light grey, respectively. Let $S \subseteq [0, 1]$ be such that $S \cap [\frac{1}{6}, \frac{1}{5}] = \emptyset$. In this case, the elements of $S \cap [\frac{4}{5}, \frac{5}{6}]$ can be rounded up to 1 instead of 0.8, which is the value given by $u^{(k)}(x)$. Indeed, items with sizes between $\frac{4}{5}$ and $\frac{5}{6}$ can only be combined with items of size lower or equal to $\frac{1}{5}$ and, for $x < \frac{1}{6}$, $u^{(k)}(x) = 0$. The corresponding loss zone

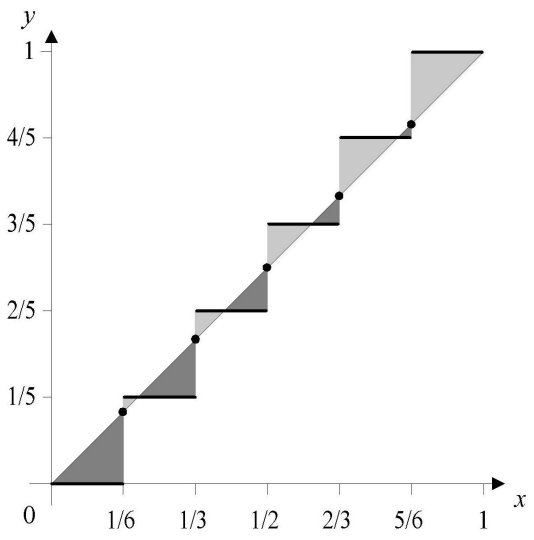

Figure 5.1: Graphical representation of $u^{(k)}$ for $k = 5$

becomes a win zone. For the same reasons, if $S \cap [\frac{1}{6}, \frac{1}{5}] = \emptyset$ and $S \cap [\frac{1}{3}, \frac{2}{5}] = \emptyset$, the elements of $S \cap [\frac{3}{5}, \frac{2}{3}]$ can be raised up to $\frac{4}{5}$ instead of $\frac{3}{5}$. Using a similar reasoning, other cases could have been identified allowing a loss zone to become a win zone. □

This observation can be used to obtain eventually stronger valid inequalities for knapsack polytopes, as long as all the coefficients of the original knapsack constraints are known, or can easily be predicted. In this thesis, we are precisely interested in the other cases, where enumerating the complete set of the knapsack constraint coefficients is impractical, and where prediction is difficult. For this reason, we will not go into further details here.

5.4 A Branch-and-Price-and-Cut Algorithm

In this section, we describe the main features of our branch-and-price-and-cut algorithm. After reviewing some aspects of the master problem initialization, we present the mapping scheme used to convert a solution from the column generation model (5.15)-(5.18) into a feasible solution of a compact arc flow model for PMP. Our branching scheme relies on this latter model. We pursue with our branching strategy, the complete description of our cutting plane procedure, and the general pricing subproblem solved at the nodes of the branch-and-bound tree.

5.4.1 Initialization

The set of admissible patterns for any practical cutting stock problem is exponential. Replicating a pattern in order to reflect a multiplicity (as is done in (5.15)-(5.18)) increases even more this already huge set. Therefore, one can only deal with a restricted

master problem, which is common with column generation reformulations, and the problem of selecting a pool of initial columns arises.

The first LP master related to (5.15)-(5.18) is initialized with an artificial column and the patterns that are in the optimal basis of the corresponding cutting stock problem. Note that this solution could certainly be improved by using one of the combination heuristics discussed in Section 5.1. The artificial column receives a high cost and coefficients equal to the right hand side of the constraints.

The algorithm used to solve the standard cutting stock problem is not of minor importance. Besides providing the optimal number of stock rolls, which is a parameter for the version of the PMP we are considering, it also provides an upper bound on the minimum number of setups, a starting incumbent. This upper bound is given by the number of different patterns that compose the optimal cutting plan, and its value may vary among different algorithms. As an illustration, in Table (5.2), we compare the bounds given by Vanderbeck in [123] with those obtained for the same instances with the algorithm for the multiple length cutting stock problem described in Chapter 3. Our algorithm gave on average slightly better results (16.5 vs 20.1). In the forthcoming computational experiments, we will resort to this algorithm.

Name	Vanderbeck [123]	MLCSP Algorithm from Chapter 3
kT03	8	6
kT05	12	11
kT01	7	6
kT02	66	20
kT04	20	16
d16p6	17	14
7p18	8	8
d33p20	17	17
12p19	15	15
d43p21	28	26
kT06	13	15
kT07	13	16
14p12	24	18
kT09	17	24
11p4	17	24
30p0	40	28
avg.	20.1	16.5

Table 5.2: Setups upper bounds with different CSP algorithms

5.4.2 Converting the LP Solution

The PMP can also be formulated as a flow problem over an acyclic digraph G, with three indexes flow variables. This is an alternative compact formulation to the assignment model discussed above. As for the multiple length cutting stock problem, we will base our branching scheme on this model, defining a comparable conversion rule to translate a LP solution to the column generation model (5.15)-(5.18) into a set of arc flows.

Graph G is composed by $W+1$ nodes, one more than the length W of the stock rolls. A unit flow going through an arc of G represents an item placed in a precise area of the roll, which appears in a pattern with a given multiplicity. The minimum number of different patters is given by the minimum weighted flow in G with additional constraints. Our arc flow model for the PMP states follows

$$\min \sum_{n=1}^{n^{max}} z^n \tag{5.41}$$

subject to

$$-\sum_{(r,s)\in A^n} x_{rs}^n + \sum_{(s,t)\in A^n} x_{st}^n = \left\{ \begin{array}{ll} z^n, & \text{for if } s=0, \\ -z^n, & \text{for } s=W, \\ 0, & \text{otherwise,} \end{array} \right. \quad n=1,\ldots,n^{max}, \tag{5.42}$$

$$\sum_{n=1}^{n^{max}} \sum_{(r,r+w_i)\in A^n} n x_{r,r+w_i}^n = b_i \ , \ i=1,\ldots,m, \tag{5.43}$$

$$x_{rs}^n \geq 0 \text{ and integer}, \ n=1,\ldots,n^{max}, \ \forall (r,s)\in A^n, \tag{5.44}$$

$$z^n \geq 0 \text{ and integer}, \ n=1,\ldots,n^{max}. \tag{5.45}$$

There is an arc set A^n for each multiplicity n from 1 up to the maximum n^{max}, as defined in (5.30). The conservation of flow occurs among the arcs of the same set A^n. It is enforced by the constraints (5.42). Variables x_{rs}^n denote the flow on arc (r,s) of A^n, while z^n, $n=1,\ldots,n^{max}$, represent the number of patterns with multiplicity n used. The loss constraint described in Section 5.2.4 is enforced by restricting in each A^n the set of loss arcs, those with a unit size. Note that although the variables of (5.15)-(5.18) are binary, the variables of the arc flow model (5.41)-(5.45) are general integer variables.

Any pattern or column in (5.15)-(5.18) can be mapped into different sets of arc flows in G, and so does an LP solution to (5.15)-(5.18). In order to associate a single path in G for each specific column of the master, so that we can easily define sets of columns with a common property to branch, we define the following mapping rule. A column with multiplicity n maps into an ordered sequence of arcs in A^n, starting at node 0, the left border of the roll. Items are converted into arcs in order of decreasing

sizes, and then, for two arcs (r, s) and (t, u) belonging to the path, we have $r > t$ implies $s - r \geq u - t$. The loss arcs are left for the end of the roll.

5.4.3 Branching Scheme

Before describing our cutting plane procedure, we introduce first the details of our branching scheme. This is motivated by the fact that cutting planes are derived not only from the constraints of the LP relaxation of (5.15)-(5.18), but also from the branching constraints.

Whenever cutting planes are not enough to close the integrality gap, we resort to branch-and-bound. Branching on the variables of the LP formulation related to (5.15)-(5.18) is inappropriate, since, to avoid column regeneration, the pricing procedure should not return the optimal solution, but rather the second best solution, or even more, depending on the depth of the node. Such a branching scheme modifies the subproblem, making it clearly intractable. A major requirement for a branching scheme is to be compatible with the subproblem. Convergence is obviously another crucial matter. In [123], Vanderbeck proposes for the PMP a list of branching rules based on hyperplanes. Even if, in practice, the author does not report any situation in which a fractional solution could not be excluded, his branching scheme does not guarantee convergence to an optimal integer solution. Branching on the arc flow variables of (5.41)-(5.45) does not induce any intractable modification to the pricing subproblem, and has the added benefit of guaranteed convergence.

Any mapping of a continuous solution to the relaxation of (5.15)-(5.18) into an equivalent solution for the LP relaxation of (5.41)-(5.45) yields, at least, one fractional variable. Here, we consider branching on one of these variables. When the solution of the LP master is fractional and the corresponding node can not be pruned by bounding, we convert the fractional solution into a set of arc flows using the mapping scheme described above. A variable x^n_{rs} with a fractional value is then selected for branching, and two nodes are created with the following branching constraints

$$x^n_{rs} \leq \lfloor x^n_{rs} \rfloor \,, \tag{5.46}$$

and

$$x^n_{rs} \geq \lceil x^n_{rs} \rceil \,. \tag{5.47}$$

These constraints are easily enforced in the LP master. All the columns with multiplicity n that map into a path with an arc (r, s) have a $+1$ coefficient in the branching constraint. With this branching strategy, we avoid the problem of symmetry, where solutions that are essentially the same appears in different branches of the tree.

When certain of these branching constraints are imposed, the LP lower bound can eventually be tightened. Consider, for example, the branching constraint $x^n_{rs} \geq lb$ with

$W - s < w_i$, for all $i = 1, \ldots, m$, *i.e.*, no item can be placed at position s. That means that part of the total waste (precisely $lb \times n(W - s)$) will surely be concentrated on patterns with multiplicity n. Hence, among the patterns with multiplicities $n' \neq n$, only those with a waste equal to or lower than

$$\left\lfloor \frac{l - lb \times n(W - s)}{n'} \right\rfloor$$

need to be generated in this node and its descendants. The patterns that are already in the master and violate this condition can be just removed.

At node w, the restricted master consists in the following LP

$$\min \sum_{p \in P'} \sum_{n \in N'_p} \lambda_{pn} \tag{5.48}$$

$$\text{subject to} \sum_{p \in P'} \sum_{n \in N'_p} n a_{ip} \lambda_{pn} = b_i, \quad i = 1, ..., m, \tag{5.49}$$

$$\sum_{p \in P'} \sum_{n \in N'_p} n \lambda_{pn} \leq z_{CSP}, \tag{5.50}$$

$$\sum_{p \in P'} \sum_{n \in N'_p} c_{ipn} \lambda_{pn} \leq 1, \ \forall i \in C, \tag{5.51}$$

$$\sum_{\{p \in P' :\ n \in N'_p\}} g^{rs}_{pn} \lambda_{pn} \leq ub^n_{rs}, \ \forall (r, s, n) \in G^w, \tag{5.52}$$

$$\sum_{\{p \in P' :\ n \in N'_p\}} g^{rs}_{pn} \lambda_{pn} \geq lb^n_{rs}, \ \forall (r, s, n) \in H^w, \tag{5.53}$$

$$0 \leq \lambda_{pn} \leq 1, \ p \in P', \ n \in N'_p. \tag{5.54}$$

The restricted set of patterns are denoted by P'. Set N'_p denotes the multiplicities for pattern p that are in the LP master. The set of branching constraints (5.46) and (5.47) enforced at a node w of the branch-and-bound tree correspond respectively to G^w and H^w. The right hand side of the branching constraints on an arc (r, s) with multiplicity n in G^w and H^w are respectively ub^n_{rs} and lb^n_{rs}. Coefficients g^{rs}_{pn} are binary. They are equal to 1 if column pn maps into a path with the arc (r, s) of A^n, and equal to 0 otherwise. Cutting planes are represented by constraints (5.51), with C being the cut set and c_{ipn} the coefficient of column pn in cut i. In the next section, cutting planes are discussed in further details.

Usually, columns with high multiplicities have fractional values in optimal LP solutions. Hence, we select for branching the variable corresponding to the leftmost arc with the higher multiplicity. Ties are broken by choosing the largest arcs. In order to favor a faster improvement of the incumbent, we used a depth-first strategy to select the next branching node to explore.

5.4.4 The Cutting Plane Procedure

Once the column generation process is complete, if the solution is not already integer and the smallest integer greater than or equal to the LP bound is strictly lower than the incumbent, we search for violated cuts before resorting to branching. Cutting planes of type (5.38), (5.39) and (5.40) are derived from three different sets of constraints:

. the constraint on the maximum number of rolls (5.50);

. the demand constraints (5.49);

. an additional waste constraint.

In fact, instead of using the exact constraint on the number of rolls as defined in (5.50), we generate cuts based on the following surrogate constraint

$$\sum_{p \in P'} \sum_{n \in N'_p} (n-1)\lambda_{pn} \leq z_{CSP} - LB^w, \tag{5.55}$$

where LB^w is the lower bound on the number of setups at a node w of the branch-and-bound tree. The surrogate constraint (5.55) results from the combination of (5.17) with

$$\sum_{p \in P'} \sum_{n \in N'_p} \lambda_{pn} \geq LB^w.$$

Note that LB^w is always the best known lower bound for node w. As a consequence, some of the cuts may be valid for that node w and its descendants, but not for the other nodes, those with an eventually smaller lower bound on the number of setups. Using this surrogate constraint may lead to stronger cuts. Indeed, the coefficients of the left hand side of (5.17) are decreased by only one unit, while its right hand side is usually decreased by a greater value. The ratio between the coefficients of the constraint and its right hand side increases, and we can expect to get cuts with greater coefficients in general. The superadditive inequalities (5.38), (5.39) and (5.40) derived from (5.55) take the following form

$$\sum_{p \in P'} \sum_{n \in N'_p} u^{(k)}\left(\frac{n-1}{z_{CSP} - LB^w}\right)\lambda_{pn} \leq 1, \ k \in \mathbb{N}, \tag{5.56}$$

$$\sum_{p \in P'} \sum_{n \in N'_p} u_1^{(\epsilon)}\left(\frac{n-1}{z_{CSP} - LB^w}\right)\lambda_{pn} \leq 1, \ \epsilon \in [0, \frac{1}{2}], \tag{5.57}$$

$$\sum_{p \in P'} \sum_{n \in N'_p} u_2^{(\epsilon)}\left(\frac{n-1}{z_{CSP} - LB^w}\right)\lambda_{pn} \leq 1, \ \epsilon \in)\frac{1}{4}, \frac{1}{2}(. \tag{5.58}$$

Using the same principle, we combine the branching constraints (5.47) at a node w with the demand constraints (5.16) to derive the following surrogate constraints

$$\sum_{p\in P'}\sum_{n'\in N'_p}(n'a_{ip}-\sum_{(r,r+w_i,n')\in H^w}g_{pn'}^{r,r+w_i})\lambda_{pn'}\leq b_i-\sum_{(r,r+w_i,n)\in H^w}\left\lceil lb^n_{r,r+w_i}\right\rceil,$$
$$i=1,\ldots,m. \quad (5.59)$$

Note that if two or more branching constraints of type (5.47) were enforced in a single arc at node w, H^w only registers the last one, the constraint with the higher right hand side. At the root node, we have $H^w=\emptyset$, and (5.59) reduces to the original demand constraint. The deeper a node is in the tree, the stronger will be the cuts. Again, constraints (5.59) are only valid on the nodes for which all the respective branching constraints are enforced. Let

$$a_{ipn'}=n'a_{ip}-\sum_{(r,r+w_i,n')\in H^w}g_{pn'}^{r,r+w_i},\qquad\text{and}\qquad b_i-\sum_{(r,r+w_i,n)\in H^w}\left\lceil lb^n_{r,r+w_i}\right\rceil,$$

for $p\in P'$, $n'\in N'_p$ and $i=1,\ldots,m$. From (5.59), we derive the following inequalities

$$\sum_{p\in P'}\sum_{n'\in N'_p}u^{(k)}\left(\frac{a_{ipn'}}{b'_i}\right)\lambda_{pn'}\leq 1,\ k\in\mathbb{N},\ i=1,...,m, \quad (5.60)$$

$$\sum_{p\in P'}\sum_{n'\in N'_p}u_1^{(\epsilon)}\left(\frac{a_{ipn'}}{b'_i}\right)\lambda_{pn'}\leq 1,\ \epsilon\in[0,\frac{1}{2}],\ i=1,...,m, \quad (5.61)$$

$$\sum_{p\in P'}\sum_{n'\in N'_p}u_2^{(\epsilon)}\left(\frac{a_{ipn'}}{b'_i}\right)\lambda_{pn'}\leq 1,\ \epsilon\in)\frac{1}{4},\frac{1}{2}(,\ i=1,...,m. \quad (5.62)$$

An additional set of cuts can be derived from the following constraint on the total waste

$$\sum_{p\in P'}\sum_{n\in N'_p}l_{pn}\lambda_{pn}\leq l.$$

Coefficients l_{pn} are the total amount of waste associated to the n copies of pattern p. This constraint is implicit in the master. It is induced by the demand constraints and the constraint on the number of rolls. However, it might help in deriving violated inequalities with the following form

$$\sum_{p\in P'}\sum_{n\in N'_p}u^{(k)}\left(\frac{l_{pn}}{l}\right)\lambda_{pn}\leq 1,\ k\in\mathbb{N}, \quad (5.63)$$

$$\sum_{p\in P'}\sum_{n\in N'_p}u_1^{(\epsilon)}\left(\frac{l_{pn}}{l}\right)\lambda_{pn}\leq 1,\ \epsilon\in[0,\frac{1}{2}], \quad (5.64)$$

$$\sum_{p\in P'}\sum_{n\in N'_p}u_2^{(\epsilon)}\left(\frac{l_{pn}}{l}\right)\lambda_{pn}\leq 1,\ \epsilon\in)\frac{1}{4},\frac{1}{2}(. \quad (5.65)$$

Among all the cuts presented above, a single cut, the most violated one, is added to the LP master in each iteration. The cutting plane procedure repeats until no more inequalities, violated in more than 0.0001, can be identified. The separation procedure is the one described in 5.3.3 for the general cuts (5.38), (5.39) and (5.40).

5.4.5 The Pricing Subproblem

The pricing subproblem solved at the root node consists in (5.24)-(5.27) with the additional loss constraint discussed in Section 5.2.4. At a node w of the branch-and-bound tree, the general reduced cost formula for a column pn is given by

$$rc_{pn} = 1 - n\left(\sum_{i=1}^{m}\pi_i a_{ip} + \rho\right) - \sum_{j\in C_1^w}\sigma_j f_j^1\left(\frac{n-1}{z_{CSP}-LB^w}\right) - \sum_{(j,i)\in C_2^w}\tau_j f_j^2\left(\frac{a_{ipn}}{b'_i}\right) - \\ - \sum_{j\in C_3^w}\upsilon_j f_j^3\left(\frac{l_{pn}}{l}\right) - \sum_{(r,s,n)\in G^w}\phi_{rs}^n g_{pn}^{rs} + \sum_{(r,s,n)\in H^w}\varphi_{rs}^n g_{pn}^{rs}. \quad (5.66)$$

According to the originating constraint of the LP master, the sets of valid cuts enforced at node w are denoted by C_1^w, C_2^w and C_3^w, for (5.56)-(5.58), (5.60)-(5.62) and (5.63)-(5.65), respectively. Vectors σ, τ and υ are the corresponding dual variables, while π, ρ, ϕ and φ denote the dual variables for the demand constraints, the constraint on the number of rolls and the branching constraints (5.52) and (5.53), respectively. Functions f_j^1, f_j^2 and f_j^3 represent the specific dual feasible function ($u^{(k)}(x)$, $u_1^{(\epsilon)}(x)$ or $u_2^{(\epsilon)}(x)$, with k and ϵ depending on the value of j) used to derive the j^{th} cut in C_1^w, C_2^w and C_3^w, respectively.

The pricing subproblem remains a nonlinear knapsack problem, which can be solved as a sequence of linear knapsack problems by fixing the multiplicity. For a multiplicity n, we can formulate the resulting pricing problem as an equivalent longest path problem in an acyclic digraph G' as follows

$$\max \sum_{(r,s)\in A'} c_{rs}x_{rs} + \sum_{j\in C_3^w}\upsilon_j f_j^3\left(\frac{loss}{l}\right) \quad (5.67)$$

subject to

$$-\sum_{(r,s)\in A'} x_{rs} + \sum_{(s,t)\in A'} x_{st} = \begin{cases} 1, & \text{for if } s=0, \\ -1, & \text{for } s=W, \\ 0, & \text{otherwise,} \end{cases} \quad (5.68)$$

$$\sum_{(r,r+w_i)\in A'} x_{r,r+w_i} \le \min\left\{\left\lfloor\frac{W}{w_i}\right\rfloor, \left\lfloor\frac{b_i}{n}\right\rfloor\right\},\ i=1,\ldots,m, \quad (5.69)$$

$$loss = W - \sum_{(r,s)\in A'\setminus A'_{loss}} x_{rs}(s-r) \le \left\lfloor\frac{l}{n}\right\rfloor, \quad (5.70)$$

$$x_{rs} \ge 0 \text{ and integer}, \forall (r,s)\in A', \quad (5.71)$$

$$loss \ge 0 \text{ and integer}. \quad (5.72)$$

Within the arc set A', the subset of loss arcs is denoted by A'_{loss}. The arc cost c_{rs} follows from (5.66).

At the root node, when there are no cutting planes in the master yet, we do not have to solve (5.67)-(5.72) for all the possible values of n. Let x^* be the knapsack representation of the optimal solution to (5.67)-(5.72), with x_i^* being the optimal number of items i in the knapsack. An optimal solution x^* for a given multiplicity n remains optimal up to

$$n^* = \min_i \left\{ \left\lfloor \frac{b_i}{x_i^*} \right\rfloor, \left\lfloor \frac{l}{W - \sum_i w_i x_i^*} \right\rfloor \right\}, \tag{5.73}$$

and therefore, (5.67)-(5.72) is solved only for $n = 1$ and the successive multiplicities given by (5.73). With branching constraints but no cuts in the master, we apply a similar scheme. When the cuts described in Section 5.4.4 are enforced in the master, we resort to a complete enumeration of the multiplicities n. Recall that the loss constraint (5.70) leads already to a significant reduction on the number of pricing subproblems that are solved in practice. Dynamic programming is finally used to solve (5.67)-(5.72).

When there are no more attractive columns, column generation stops with a proof of the optimality for the current solution of the LP master. In fact, column generation can be stopped before this optimal solution is reached if, for example, the first integer greater than the current LP solution is equal or lower than the best known lower bound for the integer PMP on that specific node. The Farley's bound [41] is also computed after each column generation iteration to eventually prune the node if the given lower bound is equal or greater than the incumbent.

5.4.6 Node Fathoming

A simple inspection of a node may help to accelerate branch-and-bound, by allowing one to anticipate the infeasibility of the corresponding problem, or by allowing to prune it by computing first a lower bound.

Consider a node w of the branch-and-bound tree with a set H^w of branching constraints of type (5.47), and let $H_1^w = \{(r, s, n) \in H^w : W - s < w_m\}$, with w_m being the size of the smallest item. If the following holds

$$\sum_{(r,s,n) \in H_1^w} n \times lb_{rs}^n \times (W - s) > l,$$

node w can be pruned, since the branching constraints force a solution with more waste than the maximum waste of the optimal CSP solution.

Based on the branching constraints at a node w, we can calculate a lower bound on the number of different patterns by two different ways. Let H_{2n}^w be the maximal set of

branching constraints (5.47) for node w, such that two pairs (r_1, s_1, n) and (r_2, s_2, n) belong to H^w_{2n} if they both belong to H^w and $s_1 - r_1 + s_2 - r_2 > W$. As a consequence, the arcs in H^w_{2n} will appear necessarily in different patterns. If

$$\sum_{n=1}^{n^{max}} \sum_{(r,s,n)\in H^w_{2n}} lb^n_{rs} \geq z_{inc}, \tag{5.74}$$

with z_{inc} being the value of the current incumbent, node w can be pruned, since it surely leads to a non improving solution. On the other hand, if

$$\sum_{n=1}^{n^{max}} \sum_{(r,s,n)\in H^w_{2n}} n \times lb^n_{rs} > z_{CSP}, \tag{5.75}$$

node w is infeasible, and can therefore be pruned in anticipation. A second lower bound can be computed as follows

$$\sum_{n=1}^{n^{max}} \max_{(r,s,n)\in H^w} lb^n_{rs},$$

since $\max_{(r,s,n)\in H^w} lb^n_{rs}$ is a lower bound on the number of patterns with multiplicity n. The node can eventually be pruned by comparing the bound with z_{inc} and z_{CSP}, as in (5.74) and (5.75) respectively.

5.5 Computational Results

Two sets of computational experiments were carried out on a 3.0 GHz Pentium IV computer with 512 MB of RAM, using CPLEX 6.5 with default settings. The first set of instances is due to Goulimis and Vance [119], and was used by Vanderbeck in [123]. Some of these instances come from real-life problems. The computational results show that our approach improves in some aspects the exact method proposed by Vanderbeck. Additionally, some experiments were performed on a set of randomly generated instances.

5.5.1 Instances from the Literature

Table 5.3 summarizes the computational results obtained with the instances used in [123]. The columns that are listed have the following interpretation

. *Name* identifies the instance;

. m is the number of different items;

. sp_{LP} is the number of subproblems solved at the root node;

- $cols_{LP}$ is the number of columns generated at the root node;
- sp_{BB} is the total number of subproblems solved at the nodes of the branch-and-bound tree, excluding the root node;
- $cols_{BB}$ is the number of columns generated during branch-and-bound;
- nod_{BB} is the number of nodes of the branch-and-bound tree, apart from the root node;
- $cuts$ is the total number of cutting planes added in the course of the algorithm;
- BBP is the initial lower bound obtained by solving the corresponding bin-packing instance;
- z_{LP}^{bc} is the LP optimum before any cut is added;
- z_{LP}^{ac} is the LP optimum after cuts are applied;
- LB is the best lower bound obtained in the course of the algorithm;
- UB is the value of the best incumbent;
- CSP is the number of different patterns given by the solution of the corresponding cutting stock problem (the values were obtained using the algorithm for the MLCSP discussed in Chapter 3);
- K is the number of stocks rolls that minimizes trim loss;
- t_{LP} is the total computing time spent at the root node;
- t_{BB} is the total time spent at the nodes of the branch-and-bound tree, after the root node;
- t_{TOT} is the total computing time.

The maximum computing time was limited to 2 hours. The algorithm solved successfully 13 of the 16 instances, while Vanderbeck only closed the optimality gap for 12 of them. However, we were still unable to find the optimal solution or prove the optimality of the incumbent for instances kT09, 11p4 and 30p0 within the limit of 2 hours. For these instances, the number of different item sizes, the size of each item compared to the length of the stock rolls and the value of the demands make the set of feasible cutting patterns relatively big. Furthermore, the optimum number of stock rolls is also high, allowing a greater diversity of patterns. Using the loss constraint described in Section 5.2.4 may help in reducing the set of cutting patterns, but for these instances this is still not enough.

	Name	m	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	*cuts*	*BBP*	z_{LP}^{bc}	z_{LP}^{ac}	*LB*	*UB*	*CSP*	*K*	t_{LP}	t_{BB}	t_{TOT}
1	kT03	7	70	64	0	0	0	46	3	4.77	5.50	6	6	6	66	0.14	0.00	0.14
2	kT05	10	55	46	157	87	65	62	4	5.65	8.00	9	9	11	47	2.80	8.38	11.17
3	kT01	5	36	35	14	14	1	57	1	2.1	2.61	3	3	6	14	0.27	0.08	0.35
4	kT02	24	121	119	93	61	26	69	13	15.93	18.00	18	18	20	66	0.62	1.17	1.78
5	kT04	16	201	166	356	281	88	113	6	6.74	7.88	9	9	16	38	2.49	11.16	13.66
6	d16p6	16	202	173	199	144	53	107	6	6.74	7.90	9	9	14	38	6.02	6.21	12.23
7	7p18	7	65	53	587	437	193	170	2	3.74	4.90	6	6	8	91	2.57	43.31	45.88
8	d33p20	23	147	146	1253	1202	114	123	5	6.18	6.70	8	8	17	29	17.08	209.03	226.11
9	12p19	12	98	96	1035	1004	63	167	2	2.89	3.90	5	5	15	23	15.84	135.59	151.43
10	d43p21	32	211	207	767	732	69	177	7	7.86	8.72	10	10	26	40	31.33	194.53	225.85
11	kT06	9	160	131	645	602	11	309	1	1.75	2.78	4	4	15	51	144.25	758.76	903.01
12	kT07	11	157	155	3458	3367	168	477	2	2.86	3.64	5	5	16	65	69.22	3393.52	3462.74
13	14p12	14	140	119	112	110	5	178	2	3.75	4.22	5	5	18	56	40.71	34.32	75.03
14	kT09	14	138	136	3656	3574	175	313	2	3.65	4.95	5	6	24	110	128.98	7071.31	7200.29
15	11p4	11	225	223	2135	2068	65	372	1	2.48	3.91	4	5	24	101	197.40	7002.88	7202.28
16	30p0	26	313	311	2051	2043	28	294	4	5.51	6.66	7	8	28	90	265.92	6935.61	7201.53
avg.		14.8	146.2	136.3	1032.4	982.9	70.3	189.6	3.8	5.16	6.27	7.1	7.3	16.5	57.8	57.85	1612.99	1670.75

Table 5.3: Computational results for instances from the literature

On average, our algorithm needs much less branching nodes. If we exclude the root, Vanderbeck needed 169 nodes, which are mainly due to the instances 7p18 and d33p20, while we only need 70.3 nodes. This is a reduction of almost 60%. Remember that we do not use any rounding heuristic with which a better incumbent may probably be found faster. In fact, we believe that the cutting plane procedure described in this chapter is more effective in terms of pruning nodes by bounding than the approach of Vanderbeck.

The LP bounds obtained after adding our cutting planes are better than the ones obtained with the cutting planes described by Vanderbeck. For kT03, we were able to strengthen the lower bound already at the root node. With 5.50, the bound can be fixed to 6 which is precisely the value of the optimal solution. The bound of 5 units obtained by Vanderbeck is improved by 10%. For most of the other instances, the LP bound is close to the next integer value (7.90 for d16p6, 4.95 for kT09, for example). On average, at the root node, the values of the LP optima are improved by 21.5%. In Table 5.4, we compare for all the instances the exact values of the LP optima obtained without any cutting plane, using our cutting planes and those proposed by Vanderbeck. With the latter, the LP bounds are improved on average by 13.6%. The lower bounds obtained with our cutting planes are always better. The improvement is indicated in percentage in the last column of the table. It goes up to 26.1%.

	Name	z^{bc}_{LP}	(1) z^{ac}_{LP}	(2) z^{ac}_{LP}	Δ (%)
1	kT03	4.77	4.99	5.50	10.22
2	kT05	5.65	7.06	8.00	13.31
3	kT01	2.10	2.46	2.61	6.10
4	kT02	15.93	17.42	18.00	3.33
5	kT04	6.74	7.79	7.88	1.16
6	d16p6	6.74	7.79	7.90	1.41
7	7p18	3.74	4.08	4.90	20.10
8	d33p20	6.18	6.5	6.70	3.08
9	12p19	2.89	3.49	3.90	11.75
10	d43p21	7.86	8.55	8.72	1.99
11	kT06	1.75	2.48	2.78	12.10
12	kT07	2.86	3.43	3.64	6.12
13	14p12	3.75	4.03	4.22	4.71
14	kT09	3.65	4.16	4.95	18.99
15	11p4	2.48	3.10	3.91	26.13
16	30p0	5.51	6.39	6.66	4.23
avg.		5.16	5.86	6.27	9.04

Table 5.4: Improvement of the LP optima with the cutting planes described in 5.4.4

5.5.2 Randomly Generated Instances

To further evaluate the performance of our algorithm, we performed a set of computational experiments on a broad range of instances randomly generated using the CUTGEN1 generator [50]. A total of 3600 instances divided in 36 groups of 100 instances were generated for different problem sizes ($m = 20$, 30 and 40), average demand ($d = 10$, 20 and 30), and relative size of the items compared to the stock lengths. The size of the smallest item (v_1) varies between 1 and 30% of the stock length, while the largest (v_2) is at most 80% of this length. We used a seed equal to 1994, and we stopped the execution after 10 minutes of branch-and-bound.

Tables 5.5 and 5.6 summarize the results obtained for the instances with $m = 20$. Column *opt* indicates the number of instances for which a proven optimal solution was found. All the instances with an average demand of 10 units per item size were solved to optimality. For the whole set, we were not able to prove the optimality of the incumbent, or find an improving one, for only 11% of the instances. The corresponding average optimality gap is not greater than 3%. The most difficult instances were those with $d = 30$. In fact, the larger the demands, the larger will be the multiplicities, and the larger will be the number of subproblems that will have to be solved. As a consequence, computing times increase naturally with the average demand, as they do with other parameters as the relative size of the smallest item, for example. Applying the cutting planes yields an improvement of 17.5% of the LP bound. The average computing time is almost 2.5 minutes.

In tables 5.7 and 5.8, we present the results for the instances with 30 different item sizes. The percentage of instances solved at optimality solved instances is now 57.6%. For these instances, the cutting planes improve the LP bound in 19.8%. The optimality gap increases to 8.9%. The average computing time is slightly greater than 6 minutes.

Tables 5.9 and 5.10 illustrate the results for the instances with $m = 40$. Only 33.5% of the instances were optimally solved, which amounts to nearly 400 instances. There is an average optimality gap of 12.1% and the computing time is almost 9 minutes.

5.6 Minimizing the Number of Different Patterns with Multiple Stock Lengths

5.6.1 Problem Formulation

With more than a single stock length, the set of cutting patterns with low trim loss becomes expectably larger. This may help in constructing a more homogeneous cutting plan, with more groups of identical patterns. Here, we consider two patterns to be identical if they have the same set of items and are cut from rolls with the same

	m	v_1	v_2	d	K	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB	CSP	opt
1	20	0.3	0.8	10	139.58	14.74	16.31	17.03	17.03	22.46	100
2	20	0.3	0.8	20	278.68	14.75	16.89	17.89	17.89	22.57	100
3	20	0.3	0.8	30	417.68	14.76	17.05	18.10	18.17	22.62	98
4	20	0.2	0.8	10	119.59	13.14	14.91	15.82	15.82	23.93	100
5	20	0.2	0.8	20	238.69	13.26	15.63	16.91	17.03	24.09	96
6	20	0.2	0.8	30	357.78	13.27	15.99	17.24	17.57	24.26	87
7	20	0.1	0.8	10	104.45	11.76	13.61	14.67	14.67	25.89	100
8	20	0.1	0.8	20	208.40	11.92	14.29	15.64	15.94	26.52	91
9	20	0.1	0.8	30	312.33	11.98	14.50	17.00	18.00	28.04	67
10	20	0.01	0.8	10	93.65	10.66	13.13	13.68	13.68	27.51	100
11	20	0.01	0.8	20	186.67	10.84	13.08	14.28	14.91	28.82	83
12	20	0.01	0.8	30	279.93	10.93	13.34	14.74	16.72	30.90	48
avg.						12.67	14.89	16.08	16.45	25.63	89.17

Table 5.5: Computational results for random instances with $m = 20$ (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{LP}	t_{BB}	t_{TOT}
1	74.23	62.02	301.26	117.55	183.42	98.01	0.51	5.47	5.98
2	132.54	107.06	650.37	282.51	343.27	225.49	2.00	31.04	33.04
3	187.42	146.26	927.28	421.88	452.30	337.09	4.64	88.42	93.06
4	94.62	80.78	421.31	174.89	244.30	131.23	0.82	11.76	12.58
5	168.66	140.95	1162.86	457.14	663.01	320.07	3.29	92.47	95.76
6	252.85	203.10	1576.88	672.87	802.83	499.64	8.65	200.33	208.98
7	108.98	96.01	812.06	323.82	487.97	184.28	1.38	33.98	35.36
8	194.33	170.00	1581.12	751.28	818.29	368.19	5.93	190.65	196.58
9	272.28	214.46	1870.53	916.74	860.70	509.65	13.74	334.08	347.81
10	122.47	108.56	833.11	459.18	405.25	175.47	3.75	48.28	52.03
11	210.38	178.64	1715.23	952.97	749.75	371.56	13.54	246.62	260.16
12	286.95	243.28	1974.86	1131.27	793.72	507.16	29.81	423.18	452.99
avg.	175.48	145.93	1152.24	555.18	567.07	310.65	7.34	142.19	149.53

Table 5.6: Computational results for random instances with $m = 20$ (b)

	m	v_1	v_2	d	K	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB	CSP	opt
1	30	0.3	0.8	10	200.76	21.04	23.09	24.01	24.03	33.16	99
2	30	0.3	0.8	20	400.96	21.18	23.98	25.17	25.35	33.11	93
3	30	0.3	0.8	30	602.40	21.24	24.36	25.44	26.02	33.20	80
4	30	0.2	0.8	10	170.96	18.49	21.13	22.19	22.30	34.15	96
5	30	0.2	0.8	20	341.48	18.63	22.10	23.12	24.18	34.34	66
6	30	0.2	0.8	30	511.99	18.66	22.49	22.98	25.35	34.82	28
7	30	0.1	0.8	10	148.71	16.77	19.42	20.46	20.86	39.36	87
8	30	0.1	0.8	20	296.94	17.01	20.40	20.91	24.02	39.24	28
9	30	0.1	0.8	30	445.07	17.08	20.69	21.26	25.67	41.79	12
10	30	0.01	0.8	10	132.80	14.95	17.36	18.28	19.26	42.76	74
11	30	0.01	0.8	20	265.10	15.25	18.41	18.65	22.36	44.53	15
12	30	0.01	0.8	30	397.30	15.45	25.03	20.54	27.14	47.82	13
avg.						17.98	21.54	21.92	23.88	38.19	57.58

Table 5.7: Computational results for random instances with $m = 30$ (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{LP}	t_{BB}	t_{TOT}
1	125.96	108.08	896.74	244.71	646.02	172.03	1.62	45.16	46.78
2	231.92	192.49	1443.36	588.72	815.09	383.45	6.84	151.04	157.88
3	322.28	264.58	1873.48	860.48	910.87	562.81	15.89	289.94	305.83
4	166.93	144.32	1247.06	419.70	813.94	241.34	2.99	86.79	89.78
5	308.63	264.33	2284.70	959.42	1271.61	531.35	13.49	353.87	367.36
6	427.59	364.21	2315.85	1176.67	1041.05	691.80	31.22	507.22	538.45
7	195.90	178.20	1753.75	751.89	1034.70	279.11	5.69	195.31	201.00
8	341.23	303.95	2667.31	1208.77	1384.54	559.11	28.90	513.71	542.61
9	475.52	405.14	2128.51	1244.35	807.56	661.15	66.10	570.79	636.89
10	212.95	194.93	2231.19	1153.95	1147.48	297.45	13.93	302.19	316.12
11	366.28	325.47	2141.58	1322.13	787.86	496.38	52.23	543.59	595.82
12	497.80	421.78	1608.09	1141.69	377.90	641.67	115.56	592.40	707.95
avg.	306.08	263.96	1882.63	922.71	919.88	459.80	29.54	346.00	375.54

Table 5.8: Computational results for random instances with $m = 30$ (b)

	m	v_1	v_2	d	K	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB	CSP	opt
1	40	0.3	0.8	10	265.02	27.25	29.75	30.65	30.85	42.98	92
2	40	0.3	0.8	20	529.24	27.42	30.76	31.65	32.69	43.45	60
3	40	0.3	0.8	30	793.74	27.45	31.19	31.77	33.68	43.95	34
4	40	0.2	0.8	10	224.85	24.03	26.98	28.00	28.65	45.10	81
5	40	0.2	0.8	20	448.90	24.35	28.41	28.92	31.48	45.34	28
6	40	0.2	0.8	30	673.25	24.38	28.85	29.13	32.26	45.53	15
7	40	0.1	0.8	10	196.18	21.66	24.80	25.35	27.18	53.20	50
8	40	0.1	0.8	20	391.54	22.04	26.19	26.43	32.65	53.00	9
9	40	0.1	0.8	30	587.08	22.34	26.70	26.83	33.20	56.92	4
10	40	0.01	0.8	10	176.90	19.46	22.39	22.64	25.62	56.77	27
11	40	0.01	0.8	20	353.12	19.79	23.51	23.66	29.87	58.93	2
12	40	0.01	0.8	30	529.44	20.03	24.21	24.35	31.27	64.47	0
avg.						23.35	26.98	27.45	30.78	50.80	33.50

Table 5.9: Computational results for random instances with $m = 40$ (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{LP}	t_{BB}	t_{TOT}
1	176.20	152.28	1451.77	425.80	1021.74	246.93	3.62	113.56	117.17
2	329.48	276.57	2340.25	979.72	1290.63	545.33	17.17	378.21	395.38
3	449.78	385.50	2165.40	1162.52	900.17	696.00	38.06	483.71	521.77
4	225.51	200.95	2152.20	683.24	1467.43	330.16	7.32	250.38	257.71
5	425.07	366.47	2467.51	1180.88	1187.95	619.27	32.96	521.75	554.71
6	591.56	503.85	1922.01	1201.19	608.47	764.56	82.56	557.29	639.85
7	266.72	245.33	2293.41	977.43	1344.97	349.63	14.52	382.67	397.19
8	475.16	422.85	2148.59	1249.08	820.62	599.52	66.30	572.92	639.23
9	650.69	557.45	1609.71	1101.08	408.75	769.46	149.77	588.70	738.47
10	302.24	274.85	2651.17	1314.74	1321.56	412.71	32.45	513.04	545.49
11	525.83	478.47	1653.81	1182.36	415.03	600.44	124.35	600.07	724.42
12	725.17	641.98	1119.96	886.39	130.94	774.15	278.54	605.04	883.58
avg.	428.62	375.55	1997.98	1028.70	909.86	559.01	70.63	463.94	534.58

Table 5.10: Computational results for random instances with $m = 40$ (b)

length. Traditionally, the cost of a solution of the standard multiple length cutting stock problem is expressed as the total length of the rolls used. Let W_{tot} denote this cost. An upper bound on the number of rolls can not be explicitly enforced as in (5.15)-(5.18), since W_{tot} can result from different combinations of rolls. Hence, if one is searching for a minimum number of different patterns without exceeding the optimal material usage, one should restrict the maximum material length used to W_{tot} units.

Let W_k, $k = 1, \ldots, K$, be the length of the stock roll k. Assume that there is a limited number of rolls of each length, which is denoted by B_k. From W_{tot}, the rolls availabilities can eventually be tightened by computing better upper bounds. A trivial upper bound on the number of rolls with length W_k is given by

$$z_k^1 = \left\lfloor \frac{W_{tot}}{W_k} \right\rfloor .$$

A stronger bound z_k^2 can be derived by solving the following problem

$$\begin{aligned} z_k^2 &= \max x_k \\ &\text{s.t. } \sum_{i=1}^{K} W_i x_i = W_{tot}, \\ &x_i \in \mathbb{N}. \end{aligned}$$

The pattern minimization problem with multiple stock lengths can be formulated using a column generation model similar to the one described in 5.2.3 as follows

$$\min \sum_{k=1}^{K} \sum_{p \in P^k} \sum_{n=1}^{ub(P_p^k)} \lambda_{pn}^k \tag{5.76}$$

subject to

$$\sum_{k=1}^{K} \sum_{p \in P^k} \sum_{n=1}^{ub(P_p^k)} n a_{ip}^k \lambda_{pn}^k = b_i, \ i = 1, \ldots, m, \tag{5.77}$$

$$\sum_{k=1}^{K} \sum_{p \in P^k} \sum_{n=1}^{ub(P_p^k)} n W_k \lambda_{pn}^k \leq W_{tot}, \tag{5.78}$$

$$\sum_{k=1}^{K} \sum_{p \in P^k} \sum_{n=1}^{ub(P_p^k)} n \lambda_{pn}^k \leq z_k^2, \ k = 1, \ldots, K, \tag{5.79}$$

$$\lambda_{pn}^k \in \{0, 1\}, \ k = 1, \ldots, K, \ p \in P^k, n = 1, \ldots, ub(P_p^k), \tag{5.80}$$

with the maximum multiplicity $ub(P_p^k)$ of a pattern p in P^k given by

$$ub(P_p^k) = \min \left\{ \min_{i=1,\ldots,m} \left\lfloor \frac{b_i}{a_{ip}^k} \right\rfloor, \left\lfloor \frac{W_{tot} - \sum_{i=1}^{m} w_i b_i}{W_k - \sum_{i=1}^{m} w_i a_{ip}^k} \right\rfloor, z_k^2 \right\} .$$

There is a nonlinear pricing subproblem for each stock length, which can again be linearized by fixing the multiplicity. Using for example a dynamic programming algorithm, the knapsack problems for different stock lengths can be solved once, as long as they share the same multiplicity.

Constraints (5.79) can be used to derive valid cutting planes with the superadditive functions $u^{(k')}(x)$, $u_1^{(\epsilon)}(x)$ and $u_2^{(\epsilon)}(x)$ discussed in 5.3.2. These cuts take the following form

$$\sum_{p\in P^k}\sum_{n=1}^{ub(P_p^k)} u^{(k')}\left(\frac{n}{z_k^2}\right)\lambda_{pn}^k \leq 1,\ k'\in\mathbb{N},\ k=1,\ldots,K, \tag{5.81}$$

$$\sum_{p\in P^k}\sum_{n=1}^{ub(P_p^k)} u_1^{(\epsilon)}\left(\frac{n}{z_k^2}\right)\lambda_{pn}^k \leq 1,\ \epsilon\in[0,\frac{1}{2}],\ k=1,\ldots,K, \tag{5.82}$$

$$\sum_{p\in P^k}\sum_{n=1}^{ub(P_p^k)} u_2^{(\epsilon)}\left(\frac{n}{z_k^2}\right)\lambda_{pn}^k \leq 1,\ \epsilon\in)\frac{1}{4},\frac{1}{2}(,\ k=1,\ldots,K. \tag{5.83}$$

These cutting planes can be derived from better surrogate constraints each time a lower bound is known on the number of rolls with a specific length. This is the case when, for example, certain branching constraints are enforced. Note that z_k^2 may be difficult to compute, and one may resort to an easier and weaker bound instead. However, the better is the bound, the stronger will be the cuts. From the other constraints of (5.76)-(5.80), cutting planes can be derived similar to the ones described in 5.4.4.

5.6.2 Computational Experiments

A set of computational experiments were conducted with a small set of instances so as to get an idea of the impact in the homogeneity of the cutting plan caused by the availability of more than a single stock length. Twenty instances were randomly generated, with 15 item sizes, and an average demand per item size of 10 units. We solved them considering the existence of a single type of stock rolls with length $W = 1000$, and the existence of two stock lengths of 1000 and 1500 units. Execution was stopped after 5 minutes of branch-and-bound. The results are reported in Table 5.11. Column *lng* represents the number of stock lengths used, and column W_{tot}, the minimum stock length (divided by 1000), which is necessary to cut the items.

A first, and obvious, observation is that more stock lengths lead to more economical cutting plans. Note that, for all the instances, the availability of the stock lengths is unlimited. Regarding the impact on the number of setups, the results obtained show consistently that the number of different patterns decreases when a greater choice of stock lengths is available. These results need to be confirmed by more extensive experiments, but they are a first confirmation of our intuition. All the instances are solved in a reasonable amount of time. The harder instances are those with two stock lengths. The computing time needed to solve them is generally far greater than for the instances with a single stock length.

	m	lng	W_{tot}	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB	t_{LP}	t_{BB}	t_{TOT}
1	15	1	102	75	58	47	24	20	74	9.93	12.00	13	13	0.66	0.52	1.18
1	15	2	88.5	82	101	2695	874	2040	303	7.29	9.00	11	11	1.69	139.03	140.73
2	15	1	90	84	61	23	12	6	72	9.16	11.00	11	11	0.82	0.25	1.07
2	15	2	74	63	116	42	75	8	81	6.42	7.84	8	8	1.46	1.18	2.65
3	14	1	132	27	23	0	0	0	11	12.00	13.00	13	13	0.13	0.00	0.13
3	14	2	99	125	116	1074	702	188	560	6.90	9.00	9	9	2.05	36.84	38.90
4	15	1	98	89	73	81	40	33	83	10.53	14.00	15	15	0.68	0.91	1.59
4	15	2	85.5	87	106	133	122	21	118	7.34	10.00	10	10	1.63	3.20	4.84
5	15	1	120	35	23	48	28	23	27	12.48	13.00	14	14	0.25	0.49	0.74
5	15	2	98.5	61	86	505	460	151	285	9.33	11.00	12	12	1.01	13.70	14.72
6	15	1	85	42	38	28	22	9	51	10.58	11.67	12	12	0.27	0.42	0.69
6	15	2	78.5	96	144	114	156	31	108	7.79	9.92	10	10	1.95	2.79	4.75
7	15	1	68	100	93	4020	454	3452	408	8.16	9.87	13	13	1.56	148.77	150.33
7	15	2	63.5	111	163	3362	2121	2577	227	6.38	7.75	9	9	3.31	301.48	304.79
8	15	1	88	81	60	157	104	44	98	9.37	11.00	12	12	0.87	1.89	2.77
8	15	2	77	81	129	721	520	296	250	7.33	8.84	9	10	2.04	28.65	30.69
9	15	1	80	44	35	15	8	8	23	10.12	11.00	12	12	0.21	0.16	0.37
9	15	2	74.5	84	122	88	95	27	95	7.00	8.79	9	9	1.50	2.27	3.78
10	15	1	94	59	41	68	38	23	54	11.58	13.00	14	14	0.51	0.68	1.19
10	15	2	84	59	81	50	42	12	54	8.25	9.00	10	10	0.95	0.86	1.81
11	14	1	93	81	62	18	9	6	61	9.26	11.00	11	11	0.65	0.17	0.82
11	14	2	81	78	124	1456	1016	857	338	6.35	8.75	9	9	1.61	78.84	80.45
12	15	1	94	50	39	55	32	20	48	11.74	13.00	14	14	0.45	0.52	0.97
12	15	2	80.5	94	117	2756	1772	1488	1372	8.09	10.00	12	12	1.59	231.77	233.36
13	15	1	94	56	50	43	36	14	57	11.25	12.50	13	13	0.50	0.50	1.00
13	15	2	79.5	69	105	5008	1979	4311	511	8.05	9.65	10	10	1.18	257.02	258.21
14	15	1	83	80	74	150	88	42	112	9.52	12.00	12	12	0.75	1.62	2.38
14	15	2	75	86	146	1095	686	460	411	6.63	8.92	11	11	2.21	59.21	61.43
15	15	1	102	55	42	205	116	62	103	10.24	12.00	12	12	0.37	2.51	2.89
15	15	2	76.5	116	125	36	38	5	82	6.98	9.00	9	9	2.26	0.88	3.14
16	14	1	150	15	14	0	0	0	0	14.00	14.00	14	14	0.08	0.00	0.08
16	14	2	112.5	91	87	108	79	21	95	7.62	9.00	9	9	1.14	1.53	2.67
17	15	1	124	27	23	8	3	4	15	13.18	13.71	14	14	0.17	0.08	0.25
17	15	2	101	57	76	210	147	87	93	9.76	10.00	11	11	0.70	3.51	4.21
18	14	1	110	70	49	78	48	23	74	10.59	12.00	13	13	0.52	0.94	1.46
18	14	2	82.5	121	119	958	497	539	194	7.08	9.00	10	10	2.69	36.95	39.64
19	14	1	99	55	38	189	107	31	205	10.83	12.00	13	13	0.46	2.70	3.16
19	14	2	82.5	69	100	20	26	6	58	7.29	8.84	9	9	1.28	0.47	1.75
20	14	1	82	111	102	37	22	18	96	8.40	11.69	12	12	1.14	0.53	1.67
20	14	2	76.5	80	134	686	918	19	556	6.44	8.73	9	9	1.83	38.60	40.44

Table 5.11: Computational results for random instances with one and two stock lengths

5.7 Conclusion

The formulation of Vanderbeck [123] for the Pattern Minimization Problem provides a poor continuous bound, and hence, it must be strengthened using additional cutting planes in order to be used in a branch-and-bound framework. In this chapter, we explored different cuts that we proved to be at least as strong as the ones used by Vanderbeck. Dual feasible functions were used for the first time for this purpose. We gave various formal proofs, showing that these functions are superadditive, and hence, that valid inequalities can be obtained with them. To get even stronger cuts, we systematically derived inequalities from various surrogate constraints, obtained by combining, for example, demand constraints with branching constraints. A branch-and-price-and-cut algorithm was developed, and tested on a set of instances from the literature. We obtained computational results slightly improved, but we were still unable to bridge the integrality gap for some of them.

The problem of minimizing setups when multiple stock lengths are available was also explored. We showed how to extend the previous approaches to this case, and we performed a set of computational experiments on some instances. With more stock lengths, one can expect to have more cutting patterns with low trim loss, which favor the construction of more homogeneous cutting plans. The first results obtained confirm this intuition.

Chapter 6

The Ordered Cutting Stock Problem

The Ordered Cutting Stock Problem was originally addressed by Ragsdale and Zobel [95], who encounter it when dealing with a door and window manufacturer. In this problem, the small items are divided into lots, and all the items belonging to the same lot must be cut from the stock rolls so that there are no items of a different lot placed among them. In practice, this constraint may arise because there is a single stack of end products near the cutting machine, or because of production requirements, like those imposed on just-in-time systems. The Ordered Cutting Stock Problem is clearly NP-complete. It combines aspects from the standard Cutting Stock Problem, and from the Traveling Salesman Problem, two problems that are well known to be NP-complete.

In this chapter, we present the first attempt to solve this problem exactly. We propose three different integer programming models for it: an assignment model, a flow model and a column generation formulation. Two families of valid cutting planes are described, along with their separation algorithms. The cuts are used to strengthen a column generation model, which is solved with a branch-and-price-and-cut algorithm. The structure of the pricing subproblem is presented, and the dynamic programming algorithm devised for its resolution is discussed. A set of random instances were generated, and computational experiments were conducted to evaluate the efficiency of our approach. The results are reported at the end of the chapter.

Keywords: Ordered Cutting Stock Problem, IP Models, Column Generation, Subtour Elimination Constraints, Comb Inequalities, Branch-and-Price-and-Cut

6.1 Introduction

Usually, cutting problems are only a part of larger production systems. Inventory control, deliveries, many are the sectors that can be affected by this kind of activities. Among the different production environments, just-in-time is surely one of the most demanding. As a consequence, cutting plans must attend to the new constraints that arise. Pattern minimization helps in reducing many costs by limiting the number of different patterns. This topic was explored in the previous chapter. Sequencing and scheduling patterns properly in the production floor are another means to control costs, or simply to ensure the respect for operational constraints. These latter problems are nowadays a true challenge for combinatorial optimization practitioners.

After a pattern is cut from the raw material, the resulting small items must be moved to a storage area, normally located close to the cutting machine. If some logical distinction is made among these items, one may think of separating them in different stacks. Here, space is an obvious restriction; the number of stacks that can be opened at the same time is therefore a bounded variable. A stack is opened when the first item of an order is cut, and remains opened until the last item is ready. Mixing items from different orders may perhaps solve the space problem, but sooner or later these items will have to be separated. With such a mixing, we can easily imagine the mess that will follow. An order can be completely defined by one or more item sizes, but it can also be composed by item sizes shared with other orders. In the first case, a stack receives items of a single size. While most of the literature in the field is devoted to this case [43, 129, 12], others generalize their approach by considering instead lots of items of different sizes [71], with the particularity that two distinct lots never have item sizes in common. The most general case consists finally in defining lots as sets of item sizes that may eventually appear in more than one lot. As far as we know, this model has been considered only by Ragsdale and Zobel in a recent paper [95], where a new problem, the Ordered Cutting Stock Problem (OCSP), is introduced. This is precisely the problem we address in this chapter.

In the OCSP, a single stack is available. Since a stack is opened for each new incoming order, lots must be processed one after the other. In this sense, to begin cutting the items of a lot, all the items from the prior scheduled lot must have been cut. We can view such a cutting process as a continuous flow of ordered lots, with lots being an indivisible entity. The sequence of lots may reflect an existing logistical policy (client priority, for example) and be fixed, total or partially, or it may be free. In this latter case, the associated optimization problem consists in finding not only the best assignment of items to the rolls, but also the best sequence of lots. Beyond the simple application to a production environment, this problem may also arise in other contexts such as truck or train loading, for example, where it is convenient for the merchandize

to be loaded without breaking the client orders. Furthermore, the orders may be larger than the capacity of a single vehicle, or wagon.

In this chapter, we propose three integer programming formulations for the OCSP, two compact models and a column generation reformulation. An exact algorithm based on column generation, branch-and-bound and cutting planes is devised to solve it. This is the first reported attempt to solve the OCSP exactly. Assignment of items to rolls, which can have different lengths, and sequencing of lots are tackled in a single phase. No a priori order for the sequence of lots is assumed. The difficulty of the "simple" cutting stock problem is hence reinforced. In fact, the OCSP combines characteristics from the common cutting stock problem with those from the Traveling Salesman Problem (TSP). We can not expect it to be easy to solve, since both are already NP-complete. Although our aim is to find optimal solutions, we will see that frequently, even when this extreme is not reached, our algorithm still provides very economical solutions, behaving as an efficient heuristic scheme.

As we mentioned above, Ragsdale and Zobel were the first to formulate this problem. They met it in an industrial context, dealing with a door and window manufacturer with an appreciable volume of activity. Their objective consists in designing an optimization algorithm able to improve both the ordering of lots and the cutting plan. And there was surely space for improvements since no rational ordering scheme had been implemented in the company, the lots being treated in a FIFO order. The need to harmonize the flow of materials in the production floor justifies the will of the company to implement a production philosophy that can lead to useful materials being discarded. Indeed, when the remaining part of a roll used to cut the items of a lot is not long enough to accommodate any of the items of the following lot, this raw material is simply thrown away. This is a defining element of the OCSP and also an immediate consequence of the just-in-time scheme for production organization, which imposes stringent constraints on inventory management and material handling.

The authors opted for a heuristic approach based on genetic algorithms. They solved both real-life data and random instances, with item sizes generated using various distributions. Chromosomes, the fitness function and crossover and mutation operators are key elements of a genetic algorithm. For the OCSP, Ragsdale and Zobel define a chromosome as a vector with information about the lots permutation and the items permutation within each lot. In their paper, the authors used the term "job" to denote what we call here a lot, and parts to denote the small items. We will keep our taxonomy, since it is closer to what is used in the cutting and packing literature. Their fitness function measures the amount of waste associated to a solution, a chromosome. Crossover and mutation operators allow one to search for an improving solution in a neighborhood. An additional heuristic is provided, the PARTCUT method, which further improves the items ordering. Comparative results are reported for the FIFO

scheme, a greedy bin-packing heuristic, the genetic algorithm applied with a fixed sequence of lots, with and without the PARTCUT heuristic, and finally the genetic algorithm with a free sequence of lots and the PARTCUT heuristic. The bin-packing heuristic already allows significant savings. The genetic algorithm by its own leads to a slight improvement only; savings become important when the PARTCUT heuristic is also used. For the real-life data, considering a variable sequence of lots conduces to a tiny reduction in the total amount of waste. For the random instances, this results in most cases in solutions with a worse level of waste. Obviously, one would expect to find at least some solutions equivalent to those obtained with a fixed order. The fact is that their genetic algorithm found difficulties in finding improving lot sequences. There is nothing odd in that, since the main complexity of the problem comes from the combination of two hard problems: sequencing and pattern construction. This is the difficulty we tried to tackle in this chapter.

The chapter is organized as follows. In the next section, we present our formulations for the OCSP. A series of cutting planes are discussed next. Two groups, namely the subtour elimination constraints and the comb inequalities, are valid inequalities that can be applied to other combinatorial problems. Our algorithm relies on the column generation formulation. We then pursue by defining the characteristics of the pricing problem. The details of our branch-and-bound algorithm are presented, and followed by the description of the rounding schemes used throughout the search tree. Computational experiments are finally reported. The characteristics of our data set are presented, and the computational results are discussed. The chapter ends with some concluding remarks.

6.2 Problem Formulations

The OCSP consists in finding a cutting plan that minimizes the total length of stocks used, various lengths of stock rolls being available, such that the items that belong to the same ordered lot are cut only when all the items of the previous lot have been cut, *i.e.*, only the remaining material from the last roll which served to cut the previous lot can be used to cut items of the next scheduled lot. We assume a bounded availability of stock rolls. Here, a distinction has to be made between "small" and "large" lots. Small lots are composed of items whose total length is smaller than at least one of the stock lengths. These lots can be completely cut from a single roll, or alternatively they may be partially cut from two different rolls. Large lots consist in orders that need more than one stock length. We will see later how the existence of two sorts of lots conditions our algorithm.

We introduce next the notation that will be used to formalize our description:

. J': set of small lots;

. J: set of lots, numbered from 1 to p, the first p' lots being the large ones, and the other $p - p'$ the small ones;

. m_j: number of different item sizes in lot j;

. K: number of different roll types;

. w_{ij}: size of item i in lot j, $w_i > w_{i+1}$ (we may have $w_{i_1 j_1} = w_{i_2 j_2}$ for $j_1 \neq j_2$);

. b_{ij}: demand for item i in lot j;

. W_k: length of roll type k (stock types are ordered from the largest W_1 to the smallest W_K);

. B_k: availability of roll type k.

The following example helps to clarify the main aspects of the OCSP.

Example 6.1 Consider an instance with four lots, one small and three large, two stock lengths of 100 and 75 units, and the following data concerning items:

lot j	m_j	(w_{ij}, b_{ij})	lot type
1	3	(65,1), (25,2), (20,2)	large
2	3	(65,1), (18,1), (5,4)	large
3	3	(65,1), (35,2), (5,5)	large
4	2	(20,1), (10,1)	small

Assume that the rolls are available in high quantities. Figure 6.1 depicts a feasible solution for the related OCSP. Lot 1 is firstly cut, followed by lot 4, lot 3 and lot 2. Since there is no imposition regarding the beginning nor the end of the sequence, an alternative ordering would be lot 2 followed by lot 3, lot 4 and lot 1. Lot 4, the small one, is completely cut from one roll, between lot 1 and lot 3, which are only partially served from this roll. In the first roll, there are 10 units of unused space, which is enough to cope with two items with 5 units of size coming from lot 2. Moving these items from the last to the first roll helps in reducing the total waste, since a roll of length 75 could be used instead of the larger roll of 100 units. However, this change produces a cycle in the lot ordering. Indeed, the resulting sequence will be lot 2, followed by lot 1, lot 4, lot 3 and lot 2 again. In these conditions, a second stack is necessary. □

At this point, a series of observations can already be made:

. a lot can only be combined with another lot in at most two rolls,

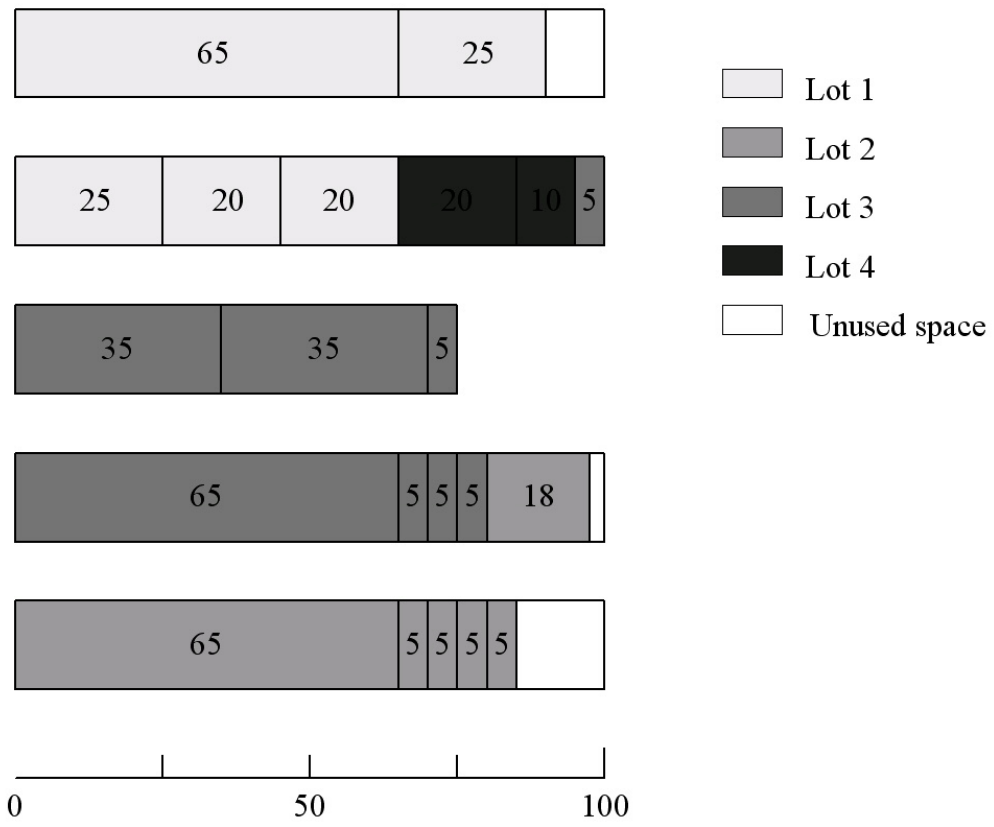

Figure 6.1: Feasible solution for the instance of Example 6.1

. there can be no more than two lots partially cut from the same roll,

. there can be any number of small lots cut from the same roll, provided that the maximum number of partially cut lots per roll is not exceeded,

. a pattern combining items from more than one lot will be cut at most once,

. lots partially cut from a roll appear necessarily at the beginning and/or the end of the roll,

. caution must be taken to avoid the creation of cycles, since they do not lead to any useful sequence.

If a cutting plan does not verify one of the conditions stated above, we will not be able to recover any feasible lot ordering from it. Whatever the sequence, there will always be a lot interrupted by some other lot, implying the opening of an additional stack.

6.2.1 An Assignment Model

In this section, we formulate the OCSP using an assignment model. Different integer and binary variables are necessary to model the assignment of items and lots to rolls. For small lots, we have to further distinguish between a complete and partial assignment

to a specific roll, since it influences the way the other lots can be assigned to this roll. The whole set of variables is defined next:

- x_{ijr} is 1 if item i of lot j is to be cut from roll r, and 0 otherwise,
- y_r is a binary variable that indicates if roll r is used or not,
- y_{jr} is 1 if lot j has been assigned to roll r, and 0 otherwise,
- z_{jr} indicates if lot j has been only partially cut from roll r. In the case of a large lot, z_{jr} is equal to y_{jr}. We will keep it anyway as it can help to clarify the formulation,
- c_{jr} indicates if the small lot j is to be completely cut from roll r,
- f_{jr} is 1 if lot j has been combined to some other lot in roll r, and 0 otherwise,
- $z_{j_1 j_2 r}$ is 1 if lots j_1 and j_2 have been both cut partially from roll r, and 0 otherwise.

To avoid obscuring even more the formulation, we present it for the case of a single stock length. The modification induced by the existence of multiple lengths follows what has been outlined in other parts of this thesis.

Consider that there are n bins of length W available. With a single type of stock rolls, our objective (6.1) becomes the minimization of the number of rolls used. The capacity or knapsack constraints are defined by inequalities (6.2), and the demand constraints by equations (6.4). Inequalities (6.3) are the definition constraints for variables y_{jr}, while (6.5) and (6.6) are those for variables c_{jr}. Using the total demand for the items of a lot in constraints (6.3), instead of a greater value, say M, strengthens the formulation. Constraints (6.7), (6.8) and (6.9) state that there can be no more than two lots partially assigned to a roll. As we pointed out in a previous observation, a lot can not "share" a roll with another lot more than twice, since this will inevitably break the lot in at least two pieces. Inequalities (6.10)-(6.12) forbid such a situation to occur. Note that if $c_{j_1 r}$ is equal to 1 for some j_1 and r, then lot j_1 will only appear in roll r. In turn, inequalities (6.13) avoid the creation of cycles, and are based on the defining constraints (6.14) for variables $z_{j_1 j_2 r}$. These variables are necessary since only the lots partially assigned to a roll can lead to a cycle. Regarding cycles, we will not go here into further detail since we devote a full section to them below.

$$\min \sum_{r=1}^{n} y_r \tag{6.1}$$

$$\text{sub. to} \sum_{j \in J} \sum_{i=1}^{m_j} w_{ij} x_{ijr} \leq W y_r, \ r = 1, \ldots, n, \tag{6.2}$$

$$\sum_{i=1}^{m_j} x_{ijr} \leq y_{jr} \sum_{i=1}^{m_j} b_{ij}, \ \forall j \in J, \ r = 1, \ldots, n, \tag{6.3}$$

$$\sum_{r=1}^{n} x_{ijr} = b_{ij},\ \forall j \in J,\ i = 1, \ldots, m_j, \tag{6.4}$$

$$\sum_{i=1}^{m_j} x_{ijr} - \sum_{i=1}^{m_j} b_{ij} \leq -\epsilon + c_{jr},\ \forall j \in J',\ r = 1, \ldots, n, \tag{6.5}$$

$$\sum_{i=1}^{m_j} x_{ijr} - \sum_{i=1}^{m_j} b_{ij} \geq (-\sum_{i=1}^{m_j} b_{ij})(1 - c_{jr}),\ \forall j \in J',\ r = 1, \ldots, n, \tag{6.6}$$

$$c_{jr} + z_{jr} = y_{jr},\ \forall j \in J',\ r = 1, \ldots, n, \tag{6.7}$$

$$z_{jr} = y_{jr},\ \forall j \in J \setminus J',\ r = 1, \ldots, n, \tag{6.8}$$

$$\sum_{j=1}^{p} z_{jr} \leq 2,\ r = 1, \ldots, n, \tag{6.9}$$

$$z_{j_1 r} + z_{j_2 r} \leq 1 + f_{j_1 r},\ \forall j_1, j_2 \in J,\ j_1 \neq j_2,\ r = 1, \ldots, n, \tag{6.10}$$

$$z_{j_1 r} + c_{j_2 r} \leq 1 + f_{j_1 r},\ \forall j_1 \in J,\ j_2 \in J',\ j_1 \neq j_2,\ r = 1, \ldots, n, \tag{6.11}$$

$$\sum_{r=1}^{n} f_{j_1 r} \leq 2,\ \forall j_1 \in J, \tag{6.12}$$

$$\sum_{i,j \in I} \sum_{r=1}^{n} z_{ijr} \leq |I| - 1,\ \forall I \subseteq J,\ 2 \leq |I| \leq p, \tag{6.13}$$

$$z_{j_1 r} + z_{j_2 r} \leq 1 + z_{j_1 j_2 r},\ j_1 = 1, ..., p, j_2 = j_1 + 1, ..., p, r = 1, ..., n, \tag{6.14}$$

$$x_{ijr} \in \mathbb{N},\ i = 1, \ldots, m_j,\ \forall j \in J,\ r = 1, \ldots, n, \tag{6.15}$$

$$y_r \in \{0, 1\},\ r = 1, \ldots, n, \tag{6.16}$$

$$z_{jr},\ y_{jr},\ f_{jr} \in \{0, 1\},\ \forall j \in J,\ r = 1, \ldots, n, \tag{6.17}$$

$$c_{jr} \in \{0, 1\},\ \forall j \in J',\ r = 1, \ldots, n, \tag{6.18}$$

$$z_{j_1 j_2 r} \in \{0, 1\},\ j_1 = 1, ..., p, j_2 = j_1 + 1, ..., p, r = 1, ..., n. \tag{6.19}$$

Usually, models that rely on assignment variables have symmetry, and formulation (6.1)-(6.19) is not an exception. Indeed, the same solution in practice may have more than one representation in this model. The number of constraints may also be important, depending on the size of the instance. Some of them can however be generated with a separation algorithm, and added to the formulation only if they are violated. This is the case of constraints (6.13). Later, a strengthened formulation obtained through a Dantzig-Wolfe decomposition of formulation (6.1)-(6.19) will be proposed, but first we present an alternative flow model for the OCSP.

6.2.2 A Flow Model

The OCSP can also be modeled as a set of flows over an acyclic digraph G with a set V of $W_1 + 1$ vertices and, fundamentally, three groups of arcs: the first represents the

inclusion of an item in an area of the roll identified by the origin and destination of the arc, the second are "transition" arcs which do not represent the occupation of space, but instead inform that a lot partially cut from a roll is followed by another lot which is also partially cut from this roll, and the third group is composed by "hybrid" arcs, which represent a transition between a lot and a small lot and the inclusion of all the items of the latter. Additionally, a fourth group of arcs could have been considered representing the usage of rolls. In this case, G would be a digraph allowing cycles and each cutting pattern would translate into a circulation flow.

In Figure 6.2, we represented the solution discussed above in Example 6.1. Only the arcs with positive flow were considered, excluding those that represent loss. The original graph G was replicated to illustrate how the conservation of flow is enforced. A set of nodes is associated to each lot. These nodes are replicated to clarify the presentation, and mainly the way the flow conservation constraints are enforced. We also included an additional set of nodes for the unique small lot of the example. All in all, there are three distinct portions in the graph of Figure 6.2, which basically represent a roll with one partial lot, with two partial lots, and with a set of complete small lots, with or without partial lots.

A pattern is translated into a set of flows according to the following rule. The partially cut lots are represented first in order of increasing indexes. The small lots which are completely cut from the roll are left for the end of the pattern. There is no way of moving between lots in the first and second portions of the depicted graph. On the contrary, transition arcs allow passing from the first to the second part of the graph. Along with the existence of two distinct portions where partially cut lots can be represented, these arcs ensure the satisfaction of the constraint which restricts to two the number of partial lots per roll. What distinguishes the arcs in the second part from those in the first part is that they are necessarily associated to a lot that follows another lot present in the roll. Cutting completely a small lot is allowed via hybrid arcs that lead to the third portion of the depicted graph. They can start in the first portion, if the complete small lots follow a unique partially cut lot, or they can start in the second portion, if the small lots come after two partially cut lots. But once the third portion is reached, there is no way of getting out of there. The only arcs that are available are hybrid arcs between two small lots, or within the same small lot if this lot is the first in the roll.

In our flow model, all these transitions are defined over a single graph with $W_1 + 1$ vertices using flow conservation constraints and different sets of arcs and flow variables. To reduce the size of our formulation, some ordering rules were applied on the lots and items within a roll. Transitions are allowed between a lot l_1 and a lot l_2 if $l_1 < l_2$, $l_1, l_2 \in J$, except for the hybrid arcs starting in the first and second portions of the graph in Figure 6.2. For the hybrid arcs in the third portion, we can also have $l_1 = l_2$.

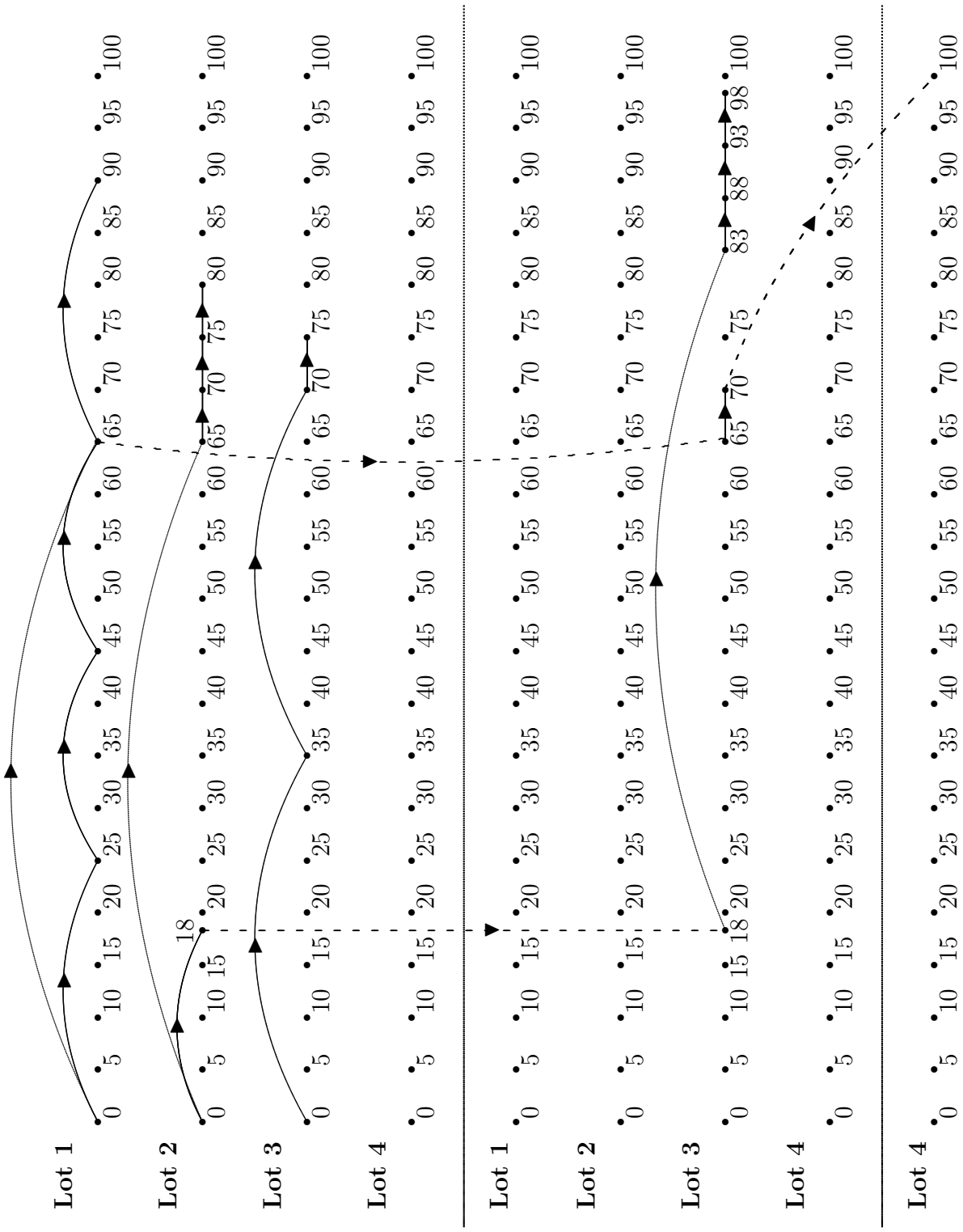

Figure 6.2: Cutting patterns presented in Example 6.1

Patterns are also represented assuming the items within a lot ordered in decreasing values of size. In our model, partial lots will always appear at the beginning of the roll. In practice, lots partially cut form a roll are always located at its extremities. However, with a complete solution for the OCSP, we can determine in a polynomial number of steps which lot goes at the beginning of the roll, and which is left for the end. The definition of the arc sets follows:

. X corresponds to the arcs in the first part of the graph in Figure 6.2. It is composed by triplets (r, s, l), r being the origin of the arc and s its destination. The difference $s - r$ must correspond to the size of an item belonging to the lot l or, alternatively, it should represent loss and have a unit length. Destination s must be less than or equal to W_1. The tail r of the arc must coincide with node 0 or with the head of another arc of X related to lot l with a length equal to or greater than $s - r$.

. Z corresponds to the arcs in the second part of the graph in Figure 6.2. Its elements have the form (r, s, l), and follow the same rules as X plus an additional one: r must be greater than or equal to the smallest item from lots l', $l' < l$. As a consequence, r will always be different from 0.

. Y is the set of transition arcs, which carry the flow from the first to the second part of the graph in Figure 6.2. Its elements have the form (s, l, l'), with $l, l' \in J$. The transition is made between the lots l and l', $l < l'$, over the same node s. The node s must be greater than or equal to the smallest item of l, and smaller than or equal to $W_1 - w_{m_{l'}l'}$, with $w_{m_{l'}}$ being the smallest item in lot l'.

. C_1 is the first set of hybrid arcs. Its elements have the form (r, s, l, l'), with $l \in J$ and $l' \in J'$ and $l \neq l'$. They represent the complete inclusion of the small lot l' after another lot l. The difference $s - r$ must be equal to the total size of the items of l'. The arcs head s must be less than or equal to W_1 and greater than or equal to the smallest item of l. The tail r must coincide with the head of another arc (r_1, r, l) of X. These arcs carry the flow from the first to the third portions of the graph in Figure 6.2.

. C_2 is the second set of hybrid arcs. Its elements have the form (r, s, l, l'), with $l \in J$, $l' \in J'$, $l \neq l'$ and $s - r = \sum_{i=1}^{m_{l'}} w_{il'}$. Additionally, the origin r of the arc must be greater than or equal to the sum of the sizes for the smallest item of l and the smallest item for a lot l_1 such that $l_1 < l$, and it must coincide with the head of an arc (r_1, r, l) of Z. The arcs of C_2 define moves between the second and third portion of the graph in Figure 6.2.

. C_3 is the last set of hybrid arcs, and have elements (r, s, l, l') with $l, l' \in J'$, $l \leq l'$ and $s - r = \sum_{i=1}^{m_{l'}} w_{il'}$. The values of l and l' are equal only for $r = 0$, and,

when they are different, r must correspond to the head of an arc with lot l as destination from C_1, C_2 or C_3. In Figure 6.2, the arcs of C_3 start and end in the third portion of the graph.

The model derived from these arc sets has considerably less symmetry than the assignment model presented above. For instances with only large lots, this symmetry is even more reduced.

The integer programming formulation for our flow model is the following:

$$\min \sum_{i=1}^{3}\sum_{k=1}^{K}\sum_{l=1}^{p} W_k g_{ik}^l \tag{6.20}$$

subject to

$$\sum_{(0,t,l)\in X} x_{0,t}^l + \sum_{(0,t,l,l')\in C_3} q_{3,0,t}^{ll'} = \sum_{i=1}^{3}\sum_{k=1}^{K}\sum_{l=1}^{p} g_{ik}^l, \tag{6.21}$$

$$- \sum_{(r,s,l)\in X} x_{rs}^l + \sum_{(s,t,l)\in X} x_{st}^l + \sum_{(s,l,l')\in Y} y_s^{ll'} + \sum_{(s,t,l,l')\in C_1} q_{1st}^{ll'} = \\ = \left\{ \begin{array}{ll} -g_{1k}^l & \text{, for } s = W_k,\ k = 1,...,K, \\ 0 & \text{, for } s \neq W_k \text{ and } s \neq 0, \end{array} \right| \ \forall l \in J, \tag{6.22}$$

$$- \sum_{(s,l',l)\in Y} y_s^{l'l} - \sum_{(r,s,l)\in Z} z_{rs}^l + \sum_{(s,t,l)\in Z} z_{st}^l + \sum_{(s,t,l,l')\in C_2} q_{2st}^{ll'} = \\ = \left\{ \begin{array}{ll} -g_{2k}^l & \text{, for } s = W_k,\ k = 1,...,K, \\ 0 & \text{, for } s \neq W_k \text{ and } s \neq 0, \end{array} \right| \ \forall l \in J, \tag{6.23}$$

$$- \sum_{(r,s,l',l)\in C_1} q_{1rs}^{l'l} - \sum_{(r,s,l',l)\in C_2} q_{2rs}^{l'l} - \sum_{(r,s,l',l)\in C_3} q_{3rs}^{l'l} + \sum_{(s,t,l,l')\in C_3} q_{3st}^{ll'} = \\ = \left\{ \begin{array}{ll} -g_{3k}^l & \text{, for } s = W_k,\ k = 1,...,K, \\ 0 & \text{, for } s \neq W_k \text{ and } s \neq 0, \end{array} \right| \ \forall l \in J', \tag{6.24}$$

$$\sum_{(s,s+w_{il},l)\in X} x_{s,s+w_{il}}^l + \sum_{(s,s+w_{il},l)\in Z} z_{s,s+w_{il}}^l + b_{il} \sum_{j=1}^{3} \sum_{(s,t,l',l)\in C_j} q_{jst}^{l'l} = b_{il}, \forall l \in J, \tag{6.25}$$

$$\sum_{(s,l',l)\in Y} y_s^{l'l} + \sum_{(s,l,l')\in Y} y_s^{ll'} + \sum_{(s,l,l')\in C_1} q_{1s}^{ll'} \leq 2,\ \forall l \in J, \tag{6.26}$$

$$\sum_{i=1}^{3}\sum_{l=1}^{p} g_{ik}^l \leq B_k,\ k = 1,...,K, \tag{6.27}$$

$$\sum_{l,l'\in I\subseteq J,(s,l,l'\in Y)} y_s^{ll'} \leq |I| - 1, \forall I \subseteq J,\ 2 \leq |I| \leq p, \tag{6.28}$$

$$x_{rs}^l \geq 0 \text{ and integer},\ \forall (r,s,l) \in X, \tag{6.29}$$

$$z_{rs}^l \geq 0 \text{ and integer},\ \forall (r,s,l) \in Z, \tag{6.30}$$

$$y_s^{ll'} \geq 0 \text{ and integer},\ \forall (s,l,l') \in Y, \tag{6.31}$$

$$q_{irs}^{ll'} \geq 0 \text{ and integer},\ i = 1,...,3,\ \forall (r,s,l,l') \in C_i, \tag{6.32}$$

$$g_{ik}^l \geq 0 \text{ and integer},\ i = 1,...,3,\ \forall l \in J. \tag{6.33}$$

All the variables in (6.20)-(6.33) are integer flow variables. Variables x_{st}^{l} and z_{st}^{l} are the flow variables for the item arcs in X and Z, respectively. Variables $y_{s}^{ll'}$ account for the flow on the transition arcs of Y, and $q_{irs}^{ll'}$ on the hybrid arcs of C_i. Even if it is not enforced explicitly, variables $y_{s}^{ll'}$ and $q_{irs}^{ll'}$ will end up as binary values. The $y_{s}^{ll'}$ variables can not be greater than 1 because of the anti-cycle constraints. In turn, since demand is satisfied exactly, variables $q_{irs}^{ll'}$ will never exceed 1. Variables g_{ik}^{l} represent the number of rolls of length W_k used, being l the last lot of the roll. In particular, g_{1k}^{l} represent the number rolls from which only the lot l is cut, g_{2k}^{l} the number of rolls used to cut lot l along with another partial lot, and g_{3k}^{l} the number of rolls used to cut completely lot l and two other lots. Clearly, in the latter case, g_{3k}^{l} will never be greater than the unit value. The constraint set (6.21)-(6.24) are the flow conservation constraints. Constraints (6.25) are the demand constraints, and (6.27) the constraints on the availability of rolls. Constraints (6.26) ensure that a lot will never appear partially in more than two different rolls. Inequalities (6.28) guarantee a lot sequence free of cycles.

6.2.3 Column Generation Reformulation

Using the Dantzig-Wolfe decomposition principle, we can obtain a LP relaxation expectably stronger than (6.1)-(6.14) keeping in the master problem the demand constraints (6.4), constraints (6.12) guaranteeing that a lot is not combined to any other lot in more than two rolls, and the anti-cycle constraints (6.13). Following this decomposition, we can formulate the integer master problem:

$$\min \sum_{k=1}^{K} \sum_{j \in P^k} W_k \lambda_{jk} \tag{6.34}$$

subject to

$$\sum_{k=1}^{K} \sum_{j \in P^k} a_{jk}^{il} \lambda_{jk} = b_{il}, \ \forall l \in J, \ i = 1, ..., m_l, \tag{6.35}$$

$$\sum_{k=1}^{K} \sum_{j \in P^k} c_{jk}^{l} \lambda_{jk} \leq 2, \ \forall l \in J, \tag{6.36}$$

$$\sum_{j \in P^k} \lambda_{jk} \leq B_k, \ k = 1, ...K, \tag{6.37}$$

$$\sum_{k=1}^{K} \sum_{j \in P^k} r_{jk}^{s} \lambda_{jk} \leq |I_s| - 1, \ \forall I_s \in S, \tag{6.38}$$

$$\lambda_{jk} \geq 0 \text{ and integer}, \ k = 1, ..., K, \ j \in P^k. \tag{6.39}$$

Each set of cutting patterns P^k, constructed from a roll of length W_k, consists in vectors $(a_{jk}^{1,1}, a_{jk}^{2,1}, ..., a_{jk}^{1,2}, ..., a_{jk}^{m_p,p})$, which indicate the number a_{jk}^{il} of items i of lot l that compose its j^{th} pattern. There are $\sum_{l=1}^{p} m_l$ elements in these vectors, as many as the number of demand constraints. For each pattern j of P^k, there is an additional vector $(c_{jk}^1, c_{jk}^2, ..., c_{jk}^p)$ of binary values, with c_{jk}^l taking the value 1 if lot l belongs partially to this pattern, and if it shares the pattern with any other lot. At most two coefficients c_{jk}^l will be positive. Note that if only one lot is to be cut partially from a roll, along with a set of complete small lots, then only one of these coefficients will be equal to 1. This time, we consider the availability constraints which are modeled via the constraints (6.37). The final group of inequalities refer to the anti-cycle constraints. Let S denote the set of all the subsets of J. For an element $I_s \in S$, the coefficient r_{jk}^s relative to column jk will be 1 if at least two lots partially cut in pattern jk belong to I_s. Otherwise, the respective coefficient will be null. Note that only the patterns with a unique lot can have a general integer frequency. The others will in fact be binary.

Figure 6.3 illustrates the restricted linear formulation obtained for the Example 6.1. Only a subset of the anti-cycle constraints is represented, namely for the lot sets $\{1, 2, 3\}$ and $\{1, 4\}$. The aim of these constraints is to avoid wrong lot sequences like 1-2-3-1, 2-1-3-2 or 1-4-1, for example. Variables $\lambda_{1,1}$, $\lambda_{2,1}$, $\lambda_{3,1}$, $\lambda_{4,1}$ and $\lambda_{1,2}$ are associated to the patterns presented in Example 6.1. Among all the variables represented, $\lambda_{1,1}$, $\lambda_{4,1}$ and $\lambda_{1,2}$ are the only true general integer variables.

The subproblem is a knapsack problem with additional constraints. It is defined by the knapsack constraints (6.2), constraints (6.3), (6.5)-(6.9) ensuring no more than two partially cut lot per roll, constraints (6.10), (6.11), (6.14) and the integrality requirements. We will go back to the subproblem later in Section 6.5.

As we have already pointed out, the number of anti-cycle constraints may be huge. Therefore, we will rather solve a relaxation of (6.34)-(6.39) where these restrictions are initially relaxed, and generate them only as they are needed. The following section is devoted to this matter. Two sections follow with other valid inequalities for the OCSP that may help strengthening the linear relaxation of our master problem. From this point forward, we will use the designation of subtour elimination constraints for these anti-cycle constraints. This is the term generally used in the TSP literature.

6.3 Subtour Elimination Constraints

6.3.1 Definition

A lot sequence can be represented as a set of flows in a complete graph $G = (V, E)$ with $p + 1$ vertices, one vertex per lot plus one extra vertex. The extra vertex will be referred to by index 0. The edges of the graph are undirected since, when a sequence is

		$\lambda_{1,1}$	$\lambda_{2,1}$	$\lambda_{3,1}$	$\lambda_{4,1}$	$\lambda_{5,1}$	$\lambda_{6,1}$	...	$\lambda_{\lvert P^1\rvert,1}$	$\lambda_{1,2}$	$\lambda_{2,2}$	...	$\lambda_{\lvert P^2\rvert,2}$		
	w_{il} =65	1				1	1							=	1
Lot 1	25	1	1			1					1			=	2
	20		2								1			=	2
	65				1									=	1
Lot 2	18			1										=	1
	5				3		1							=	4
	65			1										=	1
Lot 3	35									2				=	2
	5		1	3						1				=	5
Lot 4	20		1				1				1			=	1
	10		1			1	1				1			=	1
	l =1		1			1	1				1			$\leq$	2
	2			1			1							$\leq$	2
	3		1	1										$\leq$	2
	4					1								$\leq$	2
	W_k =100	1	1	1	1	1	1							$\leq$	500
	75									1	1			$\leq$	500
	I_s ={1,2,3}		1	1			1							$\leq$	2
	{1,4}					1								$\leq$	1
	...														

Figure 6.3: Partial LP formulation for the instance of Example 6.1

chosen, its head and tail are not specified. This was pointed out above. If a precedence relation is defined for the whole set of lots, the sequence converts into a Hamiltonian cycle of graph G. Alternatively, if the ordering of the lots lead to independent groups of lots, we translate the sequence into a set of cycles among the extra vertex and the nodes attached to each group of lots. For example, if two large lots, say l_1 and l_2, are partially cut from the same roll, and two other lots l_3 and l_4 share another roll, while the other rolls are used to cut the remaining items with a unique lot per roll, what we know is that l_2 must follow l_1, or vice versa, without any interruption to cut another lot. The same applies to l_3 and l_4. No ordering is defined between the two pairs of lots. One can determine that l_1 will be the first lot to be cut, and l_4 the last. In G, this sequence will convert into a tour starting in the extra vertex, going through the vertices corresponding to lot 1, 2, 3 and 4, and ending finally in the extra vertex. We prefer not to choose among all the hypothesis, and rather represent this sequence as the cycles $\{0, l_1, l_2, 0\}$ and $\{0, l_3, l_4, 0\}$.

Referring to formulation (6.34)-(6.39), for each pattern with a positive frequency

that includes two lots partially, there is an equal flow between the two corresponding vertices of G, representing the precedence relation between these two lots. For the small lots, the treatment is different. In fact, for some pattern, one of two situations may occur:

. all the items of the small lot belong to the pattern. In this case, it will never cause a cycle in the lot sequence, since it will never appear in any other roll, demand being satisfied exactly. As a consequence, we do not convert the pattern frequency into a flow between this small lot and some other lot present in the pattern.

. the lot belongs partially to the pattern. In this case, the remaining part of the lot will be cut from another pattern. The lot will be present in two different rolls and may link the two ends of a valid sequence. If another lot is partially included in the pattern, then the pattern frequency will be converted into a flow between it and the small lot.

The flow that converges to a vertex of $V \setminus \{0\}$ may be null or equal to 2 units. If a vertex other than 0 has a set of incident edges with a positive total flow but less than the 2 units, the gap is bridged by adding the deficit to the edge between this and the extra vertex. Examples of vertices which are somewhat isolated are those associated to small lots completely cut from a roll or to large lots that do not share any roll with another lot. Example 6.2 illustrates the details of our graph representation.

Example 6.2 Consider the instance introduced in Example 6.1. Figure 6.4 represents the graph representation for two sequences of lots. Only the arcs with a positive flow were considered. All of them have a unit flow. In (a), the sequence of Figure 6.1 is depicted. Lot 4 appears isolated, since its relation with lots 1 and 3 is not taken into account. The ends of the sequence {1,3,2} are linked to the extra vertex. An alternative ordering is represented in graph (b). It corresponds to the feasible solution of Figure 6.5. This solution defines two independent sequences, one with lots 1 and 2, and another with lots 3 and 4. The dotted lines complete the corresponding cycles with the extra vertex. In (a), linking lot 1 to lot 2 would exclude the extra vertex due to the degree constraints. This will produce a non admissible subtour. □

The only admissible cycles in G are those passing by the extra vertex. Cycles including vertices solely attached to the lots are not allowed. Let $G' = (V', E')$ be a subgraph of G, with $E' \subseteq E$, $V' \subseteq V$ and V' containing the extra vertex. These cycles can be seen as subtours of the subgraphs G' that exclude the extra vertex. As we have already seen, these subtours do not lead to any useful sequence of lots.

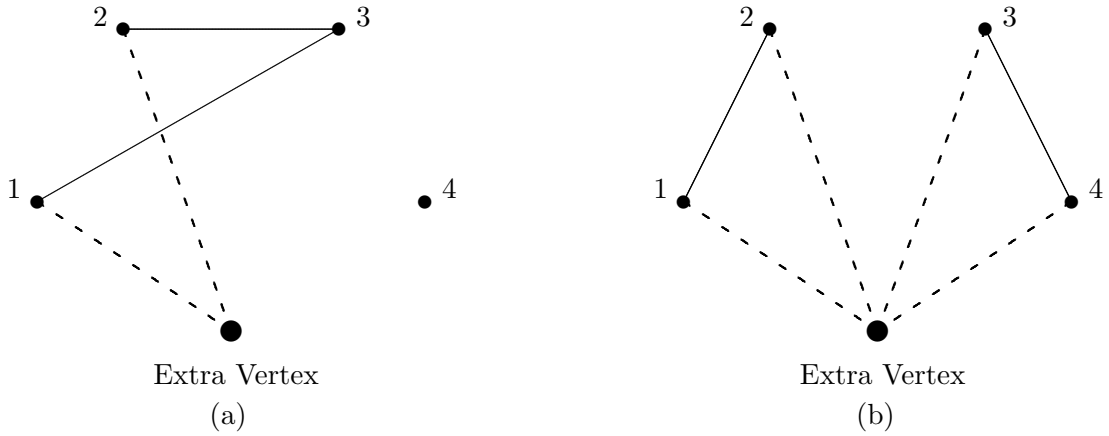

Figure 6.4: Graph representation of lot sequences (Example 6.2)

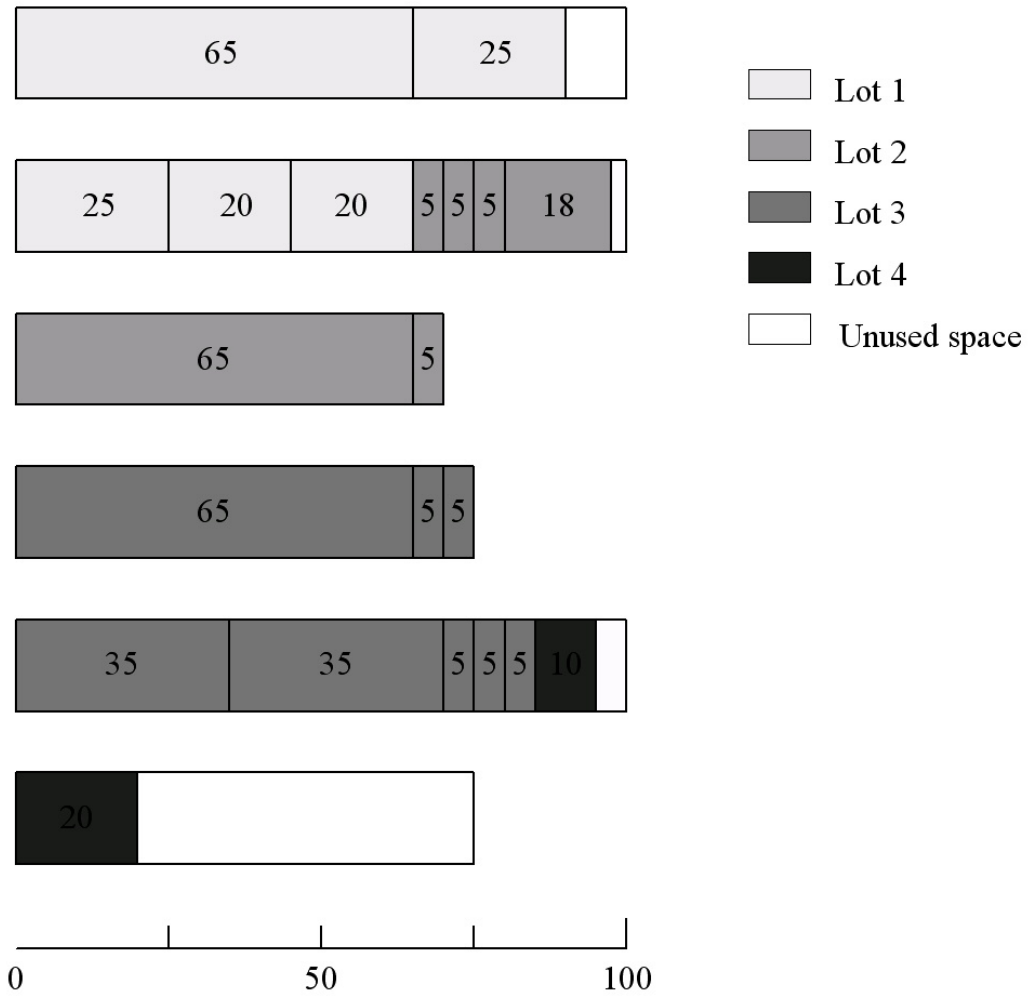

Figure 6.5: An alternative feasible solution for the instance of Example 6.1

A cycle has always a number of edges equal to its number of vertices. For all the possible G', forbidding cycles that exclude the extra vertex and, hence, have less than $|V'|$ edges, prevents the creation of invalid subtours. Let x_{ij} be the flow on edge $(i,j) \in E'$, with $i,j \in V'$ and $i < j$. The subtour elimination constraints can be stated as follows

$$\sum_{i,j \in W} x_{ij} \leq |W| - 1, \ \forall \ W \subseteq V' \setminus \{0\}, \ 2 \leq |W| \leq |V'| - 1, \ \forall \ V'. \tag{6.40}$$

These constraints correspond to (6.13), (6.28) and (6.38) in models (6.1)-(6.19), (6.20)-(6.33) and (6.34)-(6.39), respectively. Since there is an exponential number of subsets of lots, the number of subtour elimination constraints is also exponential. Enumerating them all is clearly unpractical, and therefore we will generate them only as needed. In the following section, we discuss the separation procedure used to identify violated subtour elimination constraints.

6.3.2 Separation Procedure

To search for violated subtour inequalities, we first convert the pattern frequencies given by model (6.34)-(6.39) into a set of flows x^* in G as described above. The edges $(i,j) \in E$ with $x_{ij} = 1$ are then shrunk to reduce the size of G and, hence, the size of our separation problems. From the resulting graph, we define q subgraphs $G'_s = (V'_s, E'_s)$, $s = 1, \ldots, q$, such that $\{0\} \in V'_s$, for all s, and $V'_{s_1} \cap V'_{s_2} = \{0\}$, for $s_1 \neq s_2$. The vertex sets V'_s are composed by vertices for lots connected by edges with positive flow. In the Example 6.2, for the solution represented in graph (b) of Figure 6.4, we have $V'_1 = \{0,1,2\}$ and $V'_2 = \{0,3,4\}$. The vertices of G which have no incident edge with positive flow do not belong to any of the V'_s. This is the case of vertex 4 in graph (a), Figure 6.4.

There is a subtour in G'_s if a cut-set inequality with the following form is violated by x^*:

$$\sum_{i \in W, \ j \notin W} x_{ij} \geq 2, \ \{0\} \in W, \ W \subset V'_s, \ 1 \leq |W| \leq |V'_s| - 2. \tag{6.41}$$

For ease of presentation, assume that the indexes of the vertices for lots in V'_s are redefined to vary between 1 and $|V'_s|$. The extra vertex maintains its index 0. To find a violated cut-set inequality in G'_s, one has to solve the minimum $0 - j$ cut problems

$$F_j = \min \left\{ \sum_{(i,j) \in E'_s} x_{ij} : i \in W, j \in \overline{W}, \{0,1,\ldots,j-1\} \subset W, \right.$$
$$\left. j \in \overline{W}, 1 \leq |W| \leq |V'_s| - 2 \right\}, \tag{6.42}$$

for $j = 1, \ldots, |V'_s| - 2$ [93]. If $\min_j F_j < 2$, then a subtour exists with the elements of the corresponding $\overline{W}$. In (6.42), the size of $\overline{W}$ can be of 2 elements, since we can have cycles between pairs of vertices defined by an edge between them with a flow greater than 1.

In practice, solving a maximum flow problem between the source vertex 0 and a sink vertex j leads to a solution for the minimum $0 - j$ cut problem. There are many algorithms for the former in the literature. Examples are the augmenting path algorithm from Ford and Fulkerson [45] or the preflow-push algorithms [55, 2].

6.4 Comb Inequalities

6.4.1 Definition

The comb inequalities are valid for the TSP polytope [93]. They are also valid for the Vehicle Routing Problem, for example, and can be used to strengthen our models for the OCSP. A comb is defined by a set H of vertices called the handle and t sets T_i of vertices called the teeth. The combs for a graph $G = (V, E)$ share the following properties:

. $H \cap T_i \neq \emptyset$, for $i = 1, \ldots, t$,

. $T_i \setminus H \neq \emptyset$, for $i = 1, \ldots, t$,

. $2 \leq |T_i| \leq |V| - 2$, for $i = 1, \ldots, t$,

. $T_i \cap T_j = \emptyset$, for $i \neq j$,

. $t \geq 3$ and t is odd.

For any combs of G, the following inequality holds:

$$\sum_{i,j \in H} x_{ij} + \sum_{q=1}^{t} \sum_{i,j \in T_q} x_{ij} \leq |H| + \sum_{q=1}^{t} (|T_q| - 1) - \frac{t+1}{2}. \tag{6.43}$$

In the literature, we frequently find the comb inequalities stated in another but equivalent form:

$$\sum_{i \in H,\ j \notin H} x_{ij} + \sum_{q=1}^{t} \sum_{i \in T_q,\ j \notin T_q} x_{ij} \geq 3t + 1. \tag{6.44}$$

6.4.2 Separation Procedure

The comb inequalities must hold for all the G'_s defined in Section 6.3.2. When searching for violated comb inequalities, we assume that any edge of the G'_s between the extra vertex 0 and any other edge has a flow less than or equal to the unit, and also that there is no violated subtour elimination constraint.

There is no reported exact algorithm to find general violated comb inequalities. We then resort to a heuristic procedure that is applied to each G'_s, $s = 1, \ldots, q$. The algorithm starts by defining a graph G''_s from G'_s, removing all the edges with a flow greater than $1 - \varepsilon$, where ε is a small value. A set of potential handles is then constructed with all the biconnected components of G''_s, and with the unions of at most three of these biconnected components. A biconnected component of a graph consists in a maximal set of edges such that any pair of edges appears in a common cycle. For any pair of vertices lying in a biconnected component of a graph, there are always two vertex-disjoint paths linking them. Biconnected components are found using a depth-first search. In turn, the set of candidate teeth consists in all the biconnected components of G''_s, plus the extremities of the edges (i, j) of G'_s with a flow greater than $1 - \varepsilon$. For each potential handle, we choose one by one the elements of the tooth set such that the first four properties of a valid comb stated in the previous section are never violated. We begin by the teeth with two vertices, only one in H and with a linking edge with a flow greater than $1 - \varepsilon$. Once the possibilities are exhausted, we select biconnected components of G''_s with the smaller ratio between the number of vertices and the total flow on edges that are completely within the component. At the end, if the number of teeth is even, we remove the biconnected component with the worst ratio, or if none exists, an arbitrary tooth. The corresponding comb inequality is checked, and if it is violated, we insert it in our model for the OCSP.

6.5 Pricing Columns in the Column Generation Reformulation

6.5.1 Problem Formulation

There are K pricing subproblems, one for each roll length. Each pricing subproblem is a knapsack problem where, along with the traditional capacity constraint, a restriction is imposed on the number of partial lots that can be placed in the knapsack. A set of defining constraints are also necessary in order to report into the objective function the dual values associated to the subtour elimination constraints, and those restricting to two the number of rolls where a lot can be combined to any other lot.

Let α, β, γ and δ be the vectors of dual variables related to the constraints (6.35),

(6.36), (6.37) and (6.38) from the master problem. Consider them indexed accordingly. Through the subproblem, we search for the most negative reduced cost column of the master problem. For a pattern constructed from a roll of length W_k, this cost is equal to

$$W_k - \left(\sum_{j=1}^{p} \sum_{i=1}^{m_j} \alpha_{ij} x_{ij} + \sum_{j=1}^{p} \beta_j f_j + \gamma_k + \sum_{s=1}^{|S'|} \sum_{j_1, j_2 \in I_s} \delta_s z_{j_1 j_2} \right) \tag{6.45}$$

The subproblem states as follows:

$$\max \sum_{j=1}^{p} \sum_{i=1}^{m_j} \alpha_{ij} x_{ij} + \sum_{j=1}^{p} \beta_j f_j + \sum_{s=1}^{|S'|} \sum_{j_1 j_2 \in I_s} \delta_s z_{j_1 j_2} \tag{6.46}$$

subject to

$$\sum_{j \in J} \sum_{i=1}^{m_j} w_{ij} x_{ij} \leq W_k, \tag{6.47}$$

$$\sum_{i=1}^{m_j} x_{ij} \leq y_j \sum_{i=1}^{m_j} b_{ij}, \ \forall j \in J, \tag{6.48}$$

$$\sum_{i=1}^{m_j} x_{ij} - \sum_{i=1}^{m_j} b_{ij} \leq -\epsilon + c_j, \ \forall j \in J', \tag{6.49}$$

$$\sum_{i=1}^{m_j} x_{ij} - \sum_{i=1}^{m_j} b_{ij} \geq (-\sum_{i=1}^{m_j} b_{ij})(1 - c_j), \ \forall j \in J', \tag{6.50}$$

$$c_j + z_j = y_j, \ \forall j \in J', \tag{6.51}$$

$$z_j = y_j, \ \forall j \in J \setminus J', \tag{6.52}$$

$$\sum_{j=1}^{p} z_j \leq 2, \tag{6.53}$$

$$z_{j_1} + z_{j_2} \leq 1 + f_{j_1}, \ \forall j_1, j_2 \in J, \ j_1 \neq j_2, \tag{6.54}$$

$$z_{j_1} + c_{j_2} \leq 1 + f_{j_1}, \ \forall j_1 \in J, \ j_2 \in J', \ j_1 \neq j_2, \tag{6.55}$$

$$z_{j_1} + z_{j_2} \leq 1 + z_{j_1 j_2}, \ j_1 = 1, ..., p, j_2 = j_1 + 1, ..., p, \tag{6.56}$$

$$x_{ij} \in \mathbb{N}, \ i = 1, \ldots, m_j, \ \forall j \in J \tag{6.57}$$

$$z_j, \ y_j, \ f_j \in \{0, 1\}, \ \forall j \in J, \tag{6.58}$$

$$c_j \in \{0, 1\}, \ \forall j \in J', \tag{6.59}$$

$$z_{j_1 j_2} \in \{0, 1\}, \ j_1 = 1, ..., p, j_2 = j_1 + 1, ..., p. \tag{6.60}$$

The set S' comprises only the part of the lot subsets of S considered in the master problem. Problem (6.46)-(6.60) has not the integrality property. Solving its linear relaxation does not lead necessarily to an integer solution. Additionally, observe that, for a lot j that occupies exclusively the knapsack, and whose demand is not completely satisfied from this knapsack, we can have $f_j = 0$ or $f_j = 1$. The most correct solution

would be $f_j = 0$, since lot j is not combined to any other partial lot in this pattern. In the case where there is no slack in the constraint (6.36) associated with lot j, and β_j is negative, the corresponding pattern with $f_j = 0$ will surely be generated, if it is attractive.

6.5.2 Dynamic Programming

The K subproblems are solved by running once a pseudo-polynomial dynamic programming algorithm. Its states are defined as the pairs (j_1, j_2, t, len), where j_1 and j_2 are the indexes of the partial lots in the knapsack, t is 1 or 0, if there is a complete small lot or not in the knapsack, respectively, and len is the space in the knapsack that is already occupied. When $j_1 = 0$ and $j_2 \neq 0$, the knapsack has only one partial lot. When both are 0, the knapsack has no partial lot. In this case, a value of $t = 1$ represents a knapsack filled with only complete small lots. Note that, because of constraints (6.36) and the subtour elimination constraints (and also the comb inequalities), we have to keep track of the partial lots that are in the knapsack, which are at most two. Regarding small lots, the situation is different. What we really need to register is the existence or not of one of these lots in the knapsack, since this can determine the addition of the β dual prices. For an instance with only large lots, the number of states is divided by two. Indeed, for these instances, defining a state as the pair (j_1, j_2, len) is clearly sufficient. The initial state is $(0, 0, 0, 0)$. The values of j_1 are always kept smaller than those for j_2. Considering the states ordered in increasing values for the length, the binary coefficient t and the lot indexes, the last state will then be $(p-1, p, 1, W_1)$. The total number of states is given by the following expression:

$$2 \times \left(\frac{p(p+1)}{2} + 1 \right) \times W_1 + 1$$

The cost of a state is denoted by $h(j_1, j_2, t, len)$. For the initial state, we have $h(0, 0, 0, 0) = 0$. The objective of the algorithm is to find a maximum cost solution.

We present our recurrence equation in two parts for clarity. Both depend on the ending state which corresponds respectively to a knapsack with and without any complete small lot.

$$\begin{aligned} h(j_1, j_2, 0, len) = \max_i \{\ & h(j_3, j_4, 0, len - w_{ij_2}) + \alpha_{ij_2} \mid \\ & (j_1 = 0,\ j_2 \neq 0,\ j_3 = 0 \text{ and } j_4 = 0), \text{ or} \\ & (j_1 = j_4,\ j_2 \neq 0 \text{ and } j_3 = 0), \text{ or} \\ & (j_1 = j_3 \text{ and } j_2 = j_4), \text{ or} \\ & (j_1 = 0,\ j_2 = j_4 \text{ and } j_3 = 0)\ \}, \\ \forall\ i \in \{1, \ldots, m_{j_2}\},\ & j_1, j_2, j_3 \text{ and } j_4 \in J \cup \{0\}, \end{aligned}$$

$$j_1 \leq p-1,\ j_2 \leq p,\ j_1 < j_2,\ \forall\ len \in \{w_{ij_2}, \ldots, W_1\}.$$

$$h(j_1, j_2, 1, len) = \max_{j_3} \left\{ h\left(j_1, j_2, t, len - \sum_{i=1}^{m_{j_3}} b_{ij_3} w_{ij_3}\right) + \sum_{i=1}^{m_{j_3}} b_{ij_3} \alpha_{ij_3} \;\middle|\; t = 0,\ \text{or}\ t = 1 \right\},$$

$$\forall\ j_3 \in J',\ j_1 \text{ and } j_2 \in J \cup \{0\},$$

$$j_1 \leq p-1,\ j_2 \leq p,\ j_1 < j_2,\ \forall\ len \in \left\{ \sum_{i=1}^{m_{j_3}} b_{ij_3} w_{ij_3}, \ldots, W_1 \right\}.$$

An extra term has to be added to the final cost of each state (j_1, j_2, t, len) given by the recurrence equations, in order to report the dual prices β and δ. Depending on the values for j_1, j_2 and t, a state will end up with one of the following costs:

. $h(j_1, j_2, t, len) + \beta_{j_1} + \beta_{j_2} + \sum_{s=1}^{|S'|} \delta_s st_{j_1,j_2,s}$, for $j_1 \neq 0$, $j_2 \neq 0$ and ($t = 0$ or $t = 1$);

. $h(0, j_2, 1, len) + \beta_{j_2}$, for $j_2 \neq 0$;

. $h(0, j_2, 0, len)$, for $j_2 \neq 0$;

. $h(0, 0, 1, len)$.

with $st_{j_1,j_2,s} = 1$, if j_1 and j_2 belong both to I_s, and 0 otherwise. Note that these costs could have been reported directly to the recurrence equations.

Our implementation of this dynamic programming algorithm considers a sequence of stages that can be divided in $p + (p - p')$ groups, one for each lot plus one for each small lot. Within the first p groups, there is m_l stages, if the group is attached to lot l. The last $p - p'$ groups have one stage each. In each of them, a decision is taken whether to place or not all the items of the corresponding small lot in the knapsack.

States with two partial lots are never starting points for the stages related to the first p groups. For these groups, the states to explore are only those corresponding to a knapsack with at most one partial lot. In the worst case, at most $(p+1) \times W_1 + 1$ states will have to be explored.

For the last $p - p'$ groups, the majority of the states with a cost different from ∞ (this cost stands for a state that has not been reached) have to be explored. But for each of them, there is only a binary decision to take. Furthermore, for a small lot j_3, states with the form (j_1, j_3, t, len) or (j_3, j_1, t, len), whatever the values of j_1, t and len, have not to be taken into account since demand is to be satisfied exactly, and the presence of j_3 in the list of lots signals that some of its items are already in the knapsack.

6.6 Searching for Integer Solutions with Branch-and-Bound

In this section, we discuss the characteristics of the branch-and-bound algorithm used to find the optimal integer solution for the OCSP. The branching rule adopted complicates excessively the subproblem for instances that include small lots in the lot set. Consequently, from this point forward, we will consider only instances with large lots. For those, our branching scheme is compatible with the subproblem defined in the previous section. Its complexity remains the same. We begin by giving a brief overview of the algorithm and proceed by describing how the first columns of the restricted master problem are generated. The branching scheme, search strategy and the modified subproblem are described next.

6.6.1 Algorithm Overview

The optimal integer solution for the OCSP is searched via a branch-and-price-and-cut algorithm, combining column generation and cutting planes at the branching nodes with branch-and-bound. We give an outline of our complete algorithm next.

Initially, a restricted version of the LP master problem related to (6.34)-(6.39) is solved without any of the subtour elimination constraints (6.38). Columns are generated by solving a pricing subproblem, (6.46)-(6.60) for the root node and a modified subproblem in the other nodes. The most attractive column is added to the master problem, which is then reoptimized. When there are no more attractive columns, we search for the subtour elimination constraints violated by the current solution by running the separation procedure described in Section 6.3. These violated constraints, if they exist, are inserted into the master problem, which, again, is reoptimized. The process repeats until no more violated subtour elimination constraints are found. In this case, we search for violated comb inequalities using the separation algorithm presented in Section 6.4, and repeat the column generation procedure followed by the search for violated subtour elimination constraints. Branching is done at the end of this process, when no more violated comb inequalities can be identified. Before branching, the solution of the column generation model is translated into an equivalent solution for the flow model described above. Patterns are converted into flows over item arcs and transition arcs according to the following order: lots are treated in increasing order of their indexes and items are represented within each lot from the largest to the smallest. In fact, this is the general data ordering that we have been assuming along this chapter. Note that there is no flow over hybrid arcs since we are only considering instances with large lots. Branching is done on these arcs according to the scheme we will introduce later, and branching constraints are converted back to the column generation

model following the data ordering rule. Each node is optimized following the steps already described, and the process proceeds until the optimality of the incumbent or the infeasibility of the problem are proven.

Lower bounding is used to control the number of column generation iterations. Hence, after each iteration, we calculate the Farley's bound [41], which provides a good lower bound for the master problem. If this bound is equal to or greater than the best known incumbent, column generation is stopped and the node is pruned. Besides, when the optimal solution of the LP master is equal to or lower than the best global lower bound for the integer problem, column generation is also interrupted.

6.6.2 Initialization Heuristic

The LP master problem associated with (6.34)-(6.39) is initialized with a restricted pool of columns, hence the name restricted master problem. The first column to be inserted is an artificial one ensuring the feasibility of the master in every node of the branching tree, at every stage of the column generation process. The artificial column has a high cost, so that an infeasible solution can be clearly distinguished. The step that follows consists in finding a set of columns associated to a feasible solution for the OCSP. This is done via a heuristic. The success of this step is not guaranteed since the restricted number of available rolls may cause the algorithm failure.

The objective of the heuristic is to find a good assignment of items to rolls as well as a good sequence of lots. These objectives can be considered separately or combined in a unique objective for a single algorithm. We opt for the second possibility. Our heuristic follows the principle of the well known First Fit Decreasing algorithm, and then, each item is assigned to the first opened roll where it fits. A roll is opened whenever an item, selected to be placed somewhere, does not fit in any of the already opened rolls. The rolls are opened in order of decreasing length, while lots are processed in order of increasing index. Before treating the items of a lot, all the items of the previous lot must be assigned to a roll. Within each lot, we place the items from the largest to the smallest. When an item is to be placed in a roll where there is already one or more items from a different lot, we first check if the sequence induced by this placement is free of cycles or not. In the first case, the item is assigned to the roll, and the heuristic goes on with the next item in the lot, or the first item of the following lot. In the second case, the item is not placed in the roll, and the next opened roll with enough free space is checked in turn. If there is no rolls with enough free space, a new roll is opened. At the end, the patterns are transferred one by one into the smallest roll available where they fit, freeing at the same time their original roll.

6.6.3 Branching Scheme

Branching is necessary whenever the solution of the master, after adding all the cuts, is still fractional and the first integer greater than or equal to the solution cost is strictly lower than the incumbent. To select the subset of columns to branch from the master problem, we first translate the solution of the master into a solution for the flow model using the simple rule described above. An arc (or a set of arcs) with fractional flow is then selected, and the columns whose translation into the flow model include this arc are finally the ones chosen for branching.

In order to get a balanced branching tree, we considered different levels of branching. Hence, we begin by checking the aggregated flow between two nodes induced by item arcs. Let g_{ij} denote this flow for the nodes i and j. Using the terminology introduced for the flow model, we then have

$$g_{ij} = \sum_{l \in J} x_{ij}^l + \sum_{l \in J} z_{ij}^l$$

If the current value for g_{ij} is fractional, two branching nodes are created with the following branching constraints

$$g_{ij} \leq \lfloor g_{ij} \rfloor \qquad \text{and} \qquad g_{ij} \geq \lceil g_{ij} \rceil. \tag{6.61}$$

Among the different arcs, we select the leftmost with, as a secondary criterion, the higher difference $j - i$, *i.e.*, the largest. If none of these variables is fractional, we check the x_{ij}^l variables, and finally the z_{ij}^l if all the latter are integer. The same criterion is applied for the arc selection. Based on these variables, branching constraints similar to (6.61) are enforced. The last variables to branch on if none of the latter is fractional are those associated with the transition arcs, the $y_s^{ll'}$ variables. These are binary variables, so the two branching nodes will be associated to one of the following constraints

$$y_s^{ll'} = 0 \qquad \text{and} \qquad y_s^{ll'} = 1. \tag{6.62}$$

Depending on the branching constraints that have already been enforced in a node, the second constraint may induce a cycle in the lot sequence. This situation, which can be easily anticipated, is checked and, if it is the case, the node is simply discarded. In practice, this node corresponds to an infeasible problem. The arc selected for branching is the upmost arc, *i.e.*, the one with the lowest l, located nearest from the left border of the roll (with the smallest s).

Branching on the transition arcs, or even on the x_{ij}^l or z_{ij}^l variables, does not induce any extra complication to the subproblem (6.46)-(6.60). Indeed, the states of our dynamic programming algorithm already have to register the two partial lots present in the knapsack. With a branching scheme based on hybrid arcs, these states will also have to keep track of the small lots in the knapsack. Clearly, at a certain level of the branching tree, the state space will become excessively big.

The nodes of the branch-and bound tree are selected on a best-bound basis, until the optimality gap is equal or lower than 5%. Once this point is reached, we search for an improved solution following a depth-first strategy.

6.6.4 Modified Pricing Problem

To show how the dual prices for the branching constraints are reported to the pricing subproblem, we reformulate the knapsack problem (6.46)-(6.60) as an equivalent longest path problem over an acyclic and directed graph. For this purpose, we use the set of arcs defined above for the flow model, as well as a similar set of variables. Before describing the reduced costs, we introduce first the following notation

- ρ_{ij}^{s}: dual variables for the branching constraints (6.61) on the aggregated flow in arc (i,j), with $s \in T_{1ij}^{w}$, the set of these branching constraints at node w;
- σ_{ijl}^{xs}: dual variables for the branching constraints on the x_{ij}^{l} variables, with $s \in T_{2ijl}^{w}$, for the branching node w;
- σ_{ijl}^{zs}: dual variables for the branching constraints on the z_{ij}^{l} variables, with $s \in T_{3ijl}^{w}$;
- $\tau_{ill'}^{s}$: dual variables for the branching constraints on the $y_{i}^{ll'}$ variables, with $s \in T_{4ill'}^{w}$;
- S_{w}': set of subtour elimination constraints at node w;
- δ_{s}': dual variables for the comb inequalities, with $s \in S_{w}''$;
- $f_{x,l}$: index of an item of size x in the lot l.

The reduced costs for the x_{ij}^{l}, z_{ij}^{l} and $y_{i}^{ll'}$ variables, respectively, are given by

- $\overline{c}_{ijl}^{x} = \alpha_{f_{j-i,l},l} + \sum_{s \in T_{1ij}^{w}} \rho_{ij}^{s} + \sum_{s \in T_{2ijl}^{w}} \sigma_{ijl}^{xs}$;
- $\overline{c}_{ijl}^{z} = \alpha_{f_{j-i,l},l} + \sum_{s \in T_{1ij}^{w}} \rho_{ij}^{s} + \sum_{s \in T_{2ijl}^{w}} \sigma_{ijl}^{zs}$;
- $\overline{c}_{ill'}^{y} = \sum_{s \in T_{4ill'}^{w}} \tau_{ill'}^{s} + \beta_{l} + \beta_{l'}$.

As in (6.46)-(6.60), the objective function of the modified pricing problem must account for the dual costs induced by the subtour elimination constraints (6.38). Additionally, the dual prices for the comb inequalities must also be taken into account. Hence, the formulation for the modified pricing subproblem related to a roll of length W_k stands as follows

$$\max \sum_{(i,j,l) \in X} \overline{c}_{ijl}^{x} x_{ij}^{l} + \sum_{(i,j,l) \in Z} \overline{c}_{ijl}^{z} z_{ij}^{l} + \sum_{(i,l,l') \in Y} \overline{c}_{ill'}^{y} y_{i}^{ll'} +$$

$$+\sum_{s\in S'_w}\sum_{(i,l,l')\in Y, l,l'\in I_s}\delta_s y_i^{ll'}+\sum_{s\in S''_w}\sum_{(i,l,l')\in Y}\delta'_s y_i^{ll'}cb_{ll's} \tag{6.63}$$

subject to

$$\sum_{(0,t,l)\in X} x_{0,t}^l = 1, \tag{6.64}$$

$$-\sum_{(i,j,l)\in X} x_{ij}^l+\sum_{(j,r,l)\in X} x_{jr}^l+\sum_{(i,l,l')\in Y} y_i^{ll'} = \left.\begin{cases} -g_{1k}^l & \text{, for } s=W_k,\ k=1,...,K \\ 0 & \text{, for } s\neq W_k \text{ and } s\neq 0, \end{cases}\right| ,\forall l\in J, \tag{6.65}$$

$$-\sum_{(i,l',l)\in Y} y_i^{l'l}-\sum_{(i,j,l)\in Z} z_{ij}^l+\sum_{(j,r,l)\in Z} z_{jr}^l = \left.\begin{cases} -g_{2k}^l & \text{, for } s=W_k,\ k=1,...,K \\ 0 & \text{, for } s\neq W_k \text{ and } s\neq 0, \end{cases}\right| ,\forall l\in J, \tag{6.66}$$

$$\sum_{(i,l',l)\in Y} y_i^{l'l}+\sum_{(i,l,l')\in Y} y_i^{ll'}\leq 2,\ \forall l\in J, \tag{6.67}$$

$$\sum_{l\in J} g_{1k}^l+\sum_{l\in J} g_{2k}^l = 1, \tag{6.68}$$

$$x_{ij}^l\geq 0 \text{ and integer},\ \forall (i,j,l)\in X, \tag{6.69}$$

$$z_{ij}^l\geq 0 \text{ and integer},\ \forall (i,j,l)\in Z, \tag{6.70}$$

$$y_i^{ll'}\geq 0 \text{ and integer},\ \forall (i,l,l')\in Y, \tag{6.71}$$

$$g_{ik}^l\geq 0 \text{ and integer},\ i=1,...,2,\ \forall l\in J. \tag{6.72}$$

Coefficients $cb_{l,l',s}$ are 1 if both the vertices related to lots l and l' belong to either the handle or one of the teeth of the comb s in S''_w.

The K problems (6.63)-(6.72) are solved with a single dynamic programming essentially identical to the one discussed in Section 6.5.2.

6.7 Rounding Procedures

Frequently, a solution improving the incumbent can be found quickly just by rounding the values of the current solution. This happens especially in the initial iterations of the algorithm. Hence, we devised two variants of a rounding procedure in order to improve the convergence of our branch-and-price-and-cut algorithm.

In our first rounding heuristic, all the columns associated to patterns with a single lot that have a fractional value equal to or greater than f are rounded up one by one in order of increasing indexes, so that the rolls availability is never exceeded. The columns that are not in these conditions are rounded down to the first smaller integer value. The difference between demand and the supplied items is then evaluated. Columns with a positive value are inspected in order to find the one that has items with an

excess of supply whose removal allows passing the corresponding pattern to a smaller roll. We select the column with the higher difference between the lengths of its original roll and the one for its destination roll. This step is repeated until there are no more items in excess, or the destination rolls are the same as the original ones. In the latter case, the items in excess are removed from the patterns with the lower indexes. In all the operations, the rolls availability is always taken into account. The algorithm proceeds with the placement of the remaining items. Each roll related to a column with a positive value is considered an open roll. A First Fit Decreasing strategy is used to place the items. Before assigning an item to a roll, we first check if this placement does not induce a cycle in the lot sequence.

A second procedure is used, differing only from the first in the initial steps. Before rounding up or down the columns associated to patterns with a single lot, we first check the columns with two partial lots, rounding up those with a value greater than f_1. Once more, we confirm first that this rounding does not produce any cycle in the current lot sequence. These operations are done on columns with increasing indexes.

These procedures are ran just before branching with three different values for f (0.3, 0.5 and 0.7) and two different values for f_1 (0.8 and 0.9).

6.8 Computational Experiments

6.8.1 Data Sets

We randomly generated 200 instances with five different number of lots. For each instance, the lots are composed by items whose total size is greater than the length of the larger stock roll (large lots). To guarantee that this criterion is satisfied, the instances were generated in a way such that the total number of items per lot times the smaller admissible item size (a fixed parameter) is greater than the larger roll. Four parameters were used to generate the instances. In the following tables, they are denoted by v_1 and v_2, for the sizes of the smaller and the larger items expressed as a percentage of the larger stock length, respectively, m for the average number of items per lot, and d for the average demand per item size.

Our experiments were done with 10, 15, 20, 25 and 30 lots. For each one of these values, forty instances were generated, varying the parameters m and d. The tables in the following section specify the values used for each instance. There are also two different stock lengths per instance, which belong to the set $\{60, 80, 100\}$.

6.8.2 Computational Results

Our computational experiments were carried out on a 3GHz Pentium IV computer with 512MBytes of RAM. We used CPLEX 6.5 [69] to perform some of the optimization

subroutines. The execution of the algorithm was stopped after 10 minutes spent in the branch-and-bound search tree.

The following tables list the results obtained for the 200 instances. Along with the parameters of each instance, the columns represent the value of the continuous bound before and after adding the cutting planes described in this chapter, respectively z_{LP}^{bc} and z_{LP}^{ac}, the best lower bound (LB) and the best upper bound (UB) achieved within the time limit, the number of subproblems and columns generated at the root node (sp_{LP} and $cols_{LP}$, respectively), the remaining subproblems solved in the branching tree, the number of columns generated therein and branching nodes explored (sp_{BB}, $cols_{BB}$ and nod_{BB}, respectively), the number of subtour elimination constraints and comb inequalities added ($cuts$), the time spent in initializing the master (t_{PP}), in solving the master problem and all the subproblems (t_LP and t_{SP}, respectively), the time spent with branch-and-bound (t_{BB}), and, finally, the total computing time (t_{TOT}).

We applied also the level cut described in Chapter 3. The column z_{LP}^{ac} represents the LP optimum after adding this cut, and the subtour elimination and comb inequalities. The level cut is computed using the same procedure as the one described in Chapter 3.

Only 10 instances were not solved to optimality within the time limit, 2 are from the group with 20 lots, 2 from the group with 25 lots, and 6 from the group with 30 lots. However, for all these instances, the integrality gap is very small, never greater than 0.5%. All the instances with 10 and 15 lots were solved to optimality. Not surprisingly, the larger the number of lots, the more difficult is the instance. The same happens with the number of items per lot. Beyond 30 lots, our algorithm found serious difficulties in finding an optimal integer solution, at least within a time limit of 10 minutes.

The size of the problem is directly related to the total number of lots, and item sizes per lot. For a problem with 30 lots, and 8 different item sizes per lot, for example, the demand constraints represent 240 constraints of the column generation formulation. However, the master problem is not the heavier burden. The pricing subproblem is the real bottleneck of our algorithm. Its resolution generally requires a large percentage of the total computing time. In finding improved integer solutions, the rounding procedures has proven to be very useful.

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
1	5	5	0.1	0.8	15340.00	15360.00	15360	15360
2	5	5	0.1	0.8	17005.56	17020.00	17040	17040
3	5	5	0.1	0.8	14395.83	14400.00	14440	14440
4	5	5	0.1	0.8	14946.67	14980.00	15000	15000
5	5	5	0.1	0.8	14975.91	15020.00	15040	15040
6	5	5	0.1	0.8	16035.69	16060.00	16080	16080
7	5	5	0.1	0.8	15881.00	15900.00	15900	15900
8	5	5	0.1	0.8	15990.00	16000.00	16020	16020
9	5	5	0.1	0.8	14421.67	14480.00	14480	14480
10	5	5	0.1	0.8	15895.00	15920.00	15940	15940
11	10	2	0.1	0.8	11220.00	11240.00	11240	11240
12	10	2	0.1	0.8	10325.18	10340.00	10360	10360
13	10	2	0.1	0.8	10580.00	10600.00	10600	10600
14	10	2	0.1	0.8	11805.00	11840.00	11860	11860
15	10	2	0.1	0.8	10772.89	10780.00	10800	10800
16	10	2	0.1	0.8	11467.50	11480.00	11500	11500
17	10	2	0.1	0.8	10917.50	10940.00	10960	10960
18	10	2	0.1	0.8	11360.00	11380.00	11380	11380
19	10	2	0.1	0.8	11659.00	11680.00	11680	11680
20	10	2	0.1	0.8	11394.29	11420.00	11440	11440
avg.					13319.43	13342.00	13356.00	13356.00

Table 6.1: Computational results for random instances with 10 lots (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
1	33	42	87	52	38	10	0.03	0.27	0.78	0.72	1.02
2	40	50	54	28	19	9	0.03	0.12	0.34	0.31	0.47
3	29	40	297	60	220	23	0.03	0.14	1.52	2.44	2.61
4	37	46	42	22	22	9	0.03	0.23	0.47	0.34	0.61
5	39	48	120	44	77	13	0.03	0.22	0.69	0.94	1.19
6	40	56	304	89	212	14	0.03	0.22	1.66	3.03	3.28
7	29	29	26	9	12	14	0.03	0.17	0.33	0.22	0.42
8	23	26	66	30	30	14	0.02	0.09	0.31	0.35	0.46
9	38	50	59	34	29	11	0.03	0.25	0.59	0.61	0.89
10	33	39	91	42	52	12	0.02	0.16	0.46	0.64	0.81
11	79	110	3	2	3	6	0.06	1.35	1.36	0.09	1.50
12	116	192	389	216	192	15	0.05	2.24	8.48	10.85	13.14
13	66	93	18	16	8	4	0.06	1.22	1.50	0.39	1.67
14	45	58	39	34	10	6	0.06	0.61	1.03	0.58	1.25
15	88	134	159	103	71	11	0.05	1.73	4.11	3.73	5.51
16	76	120	229	135	124	12	0.06	1.33	4.28	4.48	5.87
17	81	122	14	10	4	5	0.06	1.52	1.72	0.25	1.83
18	81	111	6	0	5	5	0.06	1.55	1.56	0.11	1.72
19	57	69	10	8	6	4	0.06	0.95	1.11	0.17	1.19
20	75	109	451	186	274	23	0.06	1.16	7.10	10.06	11.28
avg.	55.25	77.20	123.20	56.00	70.40	11.00	0.04	0.78	1.97	2.02	2.83

Table 6.2: Computational results for random instances with 10 lots (b)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
21	12	2	0.1	0.8	13688.52	13700.00	13740	13740
22	12	2	0.1	0.8	13560.00	13580.00	13600	13600
23	12	2	0.1	0.8	11868.33	11900.00	11900	11900
24	12	2	0.1	0.8	13063.33	13080.00	13100	13100
25	12	2	0.1	0.8	13214.90	13240.00	13260	13260
26	12	2	0.1	0.8	11420.00	11460.00	11500	11500
27	12	2	0.1	0.8	13144.83	13180.00	13200	13200
28	12	2	0.1	0.8	12330.00	12360.00	12380	12380
29	12	2	0.1	0.8	13760.00	13780.00	13780	13780
30	12	2	0.1	0.8	13750.00	13760.00	13780	13780
31	15	2	0.1	0.8	14606.19	14620.00	14640	14640
32	15	2	0.1	0.8	14808.15	14820.00	14820	14820
33	15	2	0.1	0.8	14535.56	14560.00	14580	14580
34	15	2	0.1	0.8	14570.00	14580.00	14600	14600
35	15	2	0.1	0.8	15977.50	16000.00	16020	16020
36	15	2	0.1	0.8	15036.58	15060.00	15080	15080
37	15	2	0.1	0.8	16000.83	16020.00	16040	16040
38	15	2	0.1	0.8	15005.88	15020.00	15060	15060
39	15	2	0.1	0.8	15170.00	15180.00	15180	15180
40	15	2	0.1	0.8	15263.33	15300.00	15340	15340
avg.					14038.70	14060.00	14080.00	14080.00

Table 6.3: Computational results for random instances with 10 lots (c)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
21	83	120	401	134	283	16	0.09	1.81	8.78	11.50	13.41
22	83	125	168	100	76	11	0.08	1.86	4.82	3.95	5.89
23	125	180	70	49	23	7	0.09	3.37	4.56	1.80	5.26
24	107	161	175	102	70	14	0.09	2.58	5.89	4.81	7.48
25	115	178	56	29	22	12	0.08	2.93	3.89	1.47	4.47
26	144	245	526	117	378	38	0.06	4.53	15.56	19.78	24.37
27	97	150	239	134	138	9	0.06	2.36	7.11	6.80	9.21
28	65	100	261	78	188	14	0.08	1.43	6.03	6.90	8.41
29	91	130	64	45	22	13	0.08	2.06	3.00	1.31	3.45
30	74	110	25	28	8	4	0.09	1.63	1.98	0.47	2.19
31	164	271	582	306	301	13	0.11	6.06	23.54	29.22	35.39
32	96	157	225	182	82	9	0.11	3.47	10.32	9.13	12.71
33	125	197	107	85	30	11	0.11	4.16	6.69	3.16	7.42
34	123	211	406	289	164	12	0.13	4.49	16.37	18.15	22.76
35	75	102	166	124	62	14	0.13	2.08	5.64	4.59	6.79
36	143	233	1216	627	631	21	0.11	4.75	44.81	80.19	85.04
37	92	153	372	188	197	18	0.13	2.91	12.86	14.28	17.31
38	142	229	346	154	206	16	0.11	5.48	15.65	15.24	20.83
39	80	144	90	105	17	4	0.13	2.63	5.19	3.01	5.76
40	114	197	204	85	104	28	0.11	3.96	8.87	7.03	11.10
avg.	106.90	169.65	284.95	148.05	150.10	14.20	0.10	3.23	10.58	12.14	15.46

Table 6.4: Computational results for random instances with 10 lots (d)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
1	4	2	0.2	0.8	7370.00	7380.00	7420	7420
2	4	2	0.2	0.8	7842.38	7860.00	7880	7880
3	4	2	0.2	0.8	7292.50	7300.00	7320	7320
4	4	2	0.2	0.8	8930.00	8940.00	8940	8940
5	4	2	0.2	0.8	8340.00	8340.00	8360	8360
6	4	2	0.2	0.8	7273.33	7280.00	7300	7300
7	4	2	0.2	0.8	7190.00	7200.00	7220	7220
8	4	2	0.2	0.8	7750.00	7800.00	7800	7800
9	4	2	0.2	0.8	7566.67	7580.00	7600	7600
10	4	2	0.2	0.8	6810.00	6820.00	6820	6820
11	6	2	0.2	0.8	11765.00	11780.00	11800	11800
12	6	2	0.2	0.8	11775.00	11780.00	11800	11800
13	6	2	0.2	0.8	12280.00	12280.00	12280	12280
14	6	2	0.2	0.8	10570.00	10580.00	10600	10600
15	6	2	0.2	0.8	11490.00	11520.00	11520	11520
16	6	2	0.2	0.8	11910.00	11920.00	11920	11920
17	6	2	0.2	0.8	11215.00	11220.00	11220	11220
18	6	2	0.2	0.8	12180.00	12200.00	12200	12200
19	6	2	0.2	0.8	11165.00	11180.00	11200	11200
20	6	2	0.2	0.8	12305.00	12320.00	12320	12320
avg.					9650.99	9664.00	9676.00	9676.00

Table 6.5: Computational results for random instances with 15 lots (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
1	59	83	170	75	82	18	0.02	0.42	1.66	1.91	2.35
2	48	71	58	42	20	5	0.03	0.30	0.61	0.49	0.82
3	56	80	118	74	44	13	0.03	0.42	1.17	1.28	1.74
4	36	52	27	17	13	2	0.03	0.24	0.35	0.14	0.41
5	41	61	21	14	8	6	0.03	0.28	0.42	0.16	0.47
6	50	67	69	37	33	14	0.03	0.36	0.81	0.66	1.05
7	52	79	44	28	14	13	0.02	0.38	0.67	0.36	0.75
8	63	77	4	1	4	4	0.02	0.41	0.42	0.03	0.46
9	53	72	73	50	28	7	0.03	0.39	0.89	0.73	1.16
10	56	75	39	25	11	14	0.03	0.42	0.69	0.38	0.83
11	75	106	305	98	187	33	0.06	1.05	4.99	5.90	7.01
12	77	96	718	122	553	63	0.06	0.97	10.01	15.37	16.40
13	57	76	11	7	7	7	0.05	0.70	0.83	0.22	0.97
14	68	102	100	56	47	9	0.06	1.02	2.33	1.76	2.83
15	66	87	20	11	11	7	0.06	0.95	1.19	0.44	1.45
16	69	93	6	3	3	1	0.06	0.84	0.87	0.11	1.02
17	78	123	268	119	147	20	0.05	1.14	4.81	5.55	6.74
18	73	89	3	0	3	6	0.06	1.05	1.08	0.09	1.20
19	76	112	255	78	166	34	0.06	1.09	4.62	4.92	6.08
20	71	87	80	50	30	17	0.06	0.89	1.87	1.17	2.12
avg.	61.20	84.40	119.45	45.35	70.55	14.65	0.04	0.67	2.02	2.08	2.79

Table 6.6: Computational results for random instances with 15 lots (b)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
21	8	2	0.2	0.8	15890.00	15900.00	15920	15920
22	8	2	0.2	0.8	14820.00	14840.00	14860	14860
23	8	2	0.2	0.8	14190.00	14220.00	14240	14240
24	8	2	0.2	0.8	15000.00	15020.00	15020	15020
25	8	2	0.2	0.8	15300.00	15320.00	15320	15320
26	8	2	0.2	0.8	15375.00	15400.00	15400	15400
27	8	2	0.2	0.8	14340.00	14360.00	14360	14360
28	8	2	0.2	0.8	14670.00	14700.00	14720	14720
29	8	2	0.2	0.8	15025.00	15040.00	15060	15060
30	8	2	0.2	0.8	13610.00	13640.00	13640	13640
31	10	2	0.2	0.8	16887.00	16900.00	16900	16900
32	10	2	0.2	0.8	16983.33	17000.00	17020	17020
33	10	2	0.2	0.8	16755.83	16780.00	16800	16800
34	10	2	0.2	0.8	17118.00	17120.00	17140	17140
35	10	2	0.2	0.8	17760.00	17780.00	17780	17780
36	10	2	0.2	0.8	16883.33	16900.00	16920	16920
37	10	2	0.2	0.8	19095.00	19120.00	19160	19160
38	10	2	0.2	0.8	17225.00	17240.00	17260	17260
39	10	2	0.2	0.8	17963.33	18000.00	18000	18000
40	10	2	0.2	0.8	18416.67	18440.00	18440	18440
avg.					16165.37	16186.00	16198.00	16198.00

Table 6.7: Computational results for random instances with 15 lots (c)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
21	76	101	276	87	174	44	0.09	1.54	6.64	6.86	8.50
22	66	98	6	5	4	5	0.11	1.53	1.62	0.16	1.79
23	73	99	141	93	58	14	0.11	1.75	4.85	3.83	5.69
24	80	109	10	0	5	12	0.09	1.67	1.81	0.24	2.00
25	75	105	113	64	49	14	0.09	1.71	4.28	3.30	5.10
26	96	115	48	35	13	12	0.09	2.07	3.01	1.13	3.29
27	75	92	14	9	4	9	0.11	1.78	2.01	0.34	2.23
28	86	112	288	95	190	36	0.09	1.81	7.61	7.88	9.78
29	87	125	218	94	119	16	0.11	2.22	7.32	6.46	8.79
30	101	154	33	19	13	8	0.09	2.42	3.09	0.88	3.39
31	88	129	445	226	223	33	0.14	3.70	21.36	23.47	27.31
32	83	124	229	94	136	28	0.14	3.62	11.92	10.53	14.30
33	114	168	624	199	418	28	0.16	5.33	30.18	34.21	39.69
34	85	125	88	58	35	8	0.14	3.09	5.93	3.53	6.76
35	99	128	30	12	14	9	0.14	3.59	4.47	1.15	4.89
36	105	168	1150	280	838	65	0.14	4.64	52.05	67.04	71.82
37	80	100	693	129	502	92	0.16	2.60	22.40	27.86	30.62
38	103	160	554	256	300	39	0.16	4.09	26.00	30.38	34.63
39	78	115	30	20	13	6	0.14	2.82	3.76	1.25	4.21
40	94	136	13	2	6	8	0.14	3.61	3.97	0.53	4.28
avg.	87.20	123.15	250.15	88.85	155.70	24.30	0.12	2.78	11.21	11.55	14.45

Table 6.8: Computational results for random instances with 15 lots (d)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
1	3	2	0.2	0.8	8685.00	8700.00	8700	8700
2	3	2	0.2	0.8	8610.00	8620.00	8640	8640
3	3	2	0.2	0.8	8072.50	8080.00	8080	8080
4	3	2	0.2	0.8	7575.00	7580.00	7600	7600
5	3	2	0.2	0.8	7770.00	7780.00	7780	7780
6	3	2	0.2	0.8	8620.00	8640.00	8640	8640
7	3	2	0.2	0.8	8805.00	8820.00	8820	8820
8	3	2	0.2	0.8	8355.00	8380.00	8380	8380
9	3	2	0.2	0.8	7770.00	7780.00	7800	7800
10	3	2	0.2	0.8	7510.00	7520.00	7540	7540
11	5	2	0.2	0.8	12170.00	12180.00	12200	12200
12	5	2	0.2	0.8	12770.00	12780.00	12800	12800
13	5	2	0.2	0.8	13173.33	13180.00	13180	13180
14	5	2	0.2	0.8	12556.67	12580.00	12580	12580
15	5	2	0.2	0.8	13150.00	13160.00	13180	13180
16	5	2	0.2	0.8	11875.00	11880.00	11900	11900
17	5	2	0.2	0.8	12005.00	12020.00	12020	12020
18	5	2	0.2	0.8	12640.00	12660.00	12660	12660
19	5	2	0.2	0.8	13030.00	13060.00	13080	13080
20	5	2	0.2	0.8	12430.00	12480.00	12480	12480
avg.					10378.63	10394.00	10403.00	10403.00

Table 6.9: Computational results for random instances with 20 lots (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
1	56	64	45	21	23	9	0.03	0.45	0.83	0.55	1.03
2	62	76	230	66	164	22	0.03	0.55	2.60	3.09	3.66
3	56	67	12	5	4	7	0.03	0.50	0.58	0.14	0.67
4	57	64	11	7	4	6	0.02	0.50	0.58	0.13	0.64
5	67	79	4	0	2	4	0.03	0.55	0.56	0.05	0.62
6	63	80	18	5	10	9	0.03	0.55	0.75	0.23	0.81
7	30	38	5	0	3	4	0.03	0.28	0.30	0.06	0.38
8	72	82	98	31	60	19	0.03	0.63	1.27	1.00	1.66
9	54	78	23	12	9	4	0.03	0.56	0.79	0.28	0.87
10	61	78	270	76	188	21	0.03	0.58	3.03	3.85	4.46
11	103	145	19	13	4	11	0.08	2.53	2.91	0.47	3.08
12	91	119	44	22	18	18	0.06	1.92	2.65	0.97	2.95
13	113	133	59	32	24	12	0.08	2.67	3.88	1.60	4.35
14	95	123	15	3	10	5	0.08	2.19	2.47	0.47	2.73
15	91	123	22	10	10	6	0.06	2.02	2.43	0.63	2.71
16	90	136	657	211	427	51	0.08	2.33	19.66	25.54	27.95
17	71	99	33	19	11	13	0.08	1.78	2.55	1.09	2.95
18	87	112	32	14	14	16	0.08	1.90	2.55	0.85	2.83
19	71	92	59	35	19	19	0.06	1.56	2.77	1.53	3.15
20	94	130	38	26	20	9	0.08	2.09	2.92	1.15	3.31
avg.	74.20	95.90	84.70	30.40	51.20	13.25	0.05	1.31	2.80	2.18	3.54

Table 6.10: Computational results for random instances with 20 lots (b)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
21	6	2	0.2	0.8	16030.00	16040.00	16040	16040
22	6	2	0.2	0.8	16680.00	16700.00	16700	16700
23	6	2	0.2	0.8	15810.00	15820.00	15840	15840
24	6	2	0.2	0.8	15705.00	15720.00	15720	15720
25	6	2	0.2	0.8	15330.00	15340.00	15340	15340
26	6	2	0.2	0.8	15960.00	15980.00	15980	15980
27	6	2	0.2	0.8	15127.00	15140.00	15140	15140
28	6	2	0.2	0.8	14895.00	14920.00	14920	14920
29	6	2	0.2	0.8	14720.00	14740.00	14740	14740
30	6	2	0.2	0.8	14660.00	14680.00	14680	14680
31	8	2	0.2	0.8	20330.00	20340.00	20360	20360
32	8	2	0.2	0.8	19190.00	19220.00	19240	19240
33	8	2	0.2	0.8	19375.00	19400.00	19400	19400
34	8	2	0.2	0.8	20499.58	20520.00	20520	20540
35	8	2	0.2	0.8	19450.00	19460.00	19460	19460
36	8	2	0.2	0.8	19430.00	19440.00	19440	19440
37	8	2	0.2	0.8	19940.00	19940.00	19960	19960
38	8	2	0.2	0.8	19870.00	19880.00	19900	19900
39	8	2	0.2	0.8	20497.50	20520.00	20520	20520
40	8	2	0.2	0.8	18923.33	18960.00	18960	18980
avg.					17621.12	17638.00	17643.00	17645.00

Table 6.11: Computational results for random instances with 20 lots (c)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
21	73	95	21	11	9	5	0.11	2.30	2.91	0.78	3.19
22	104	144	41	16	23	5	0.09	2.97	4.07	1.55	4.61
23	78	108	16	14	2	7	0.09	2.30	2.72	0.49	2.87
24	82	120	89	47	36	13	0.11	2.58	5.08	3.11	5.80
25	82	100	25	14	9	11	0.11	2.53	3.20	0.89	3.53
26	133	164	33	14	14	17	0.11	3.80	4.57	1.16	5.06
27	117	167	10	5	5	8	0.11	4.08	4.39	0.39	4.58
28	109	135	761	231	489	79	0.11	3.19	23.70	32.27	35.57
29	109	141	13	6	6	13	0.11	3.55	3.83	0.52	4.17
30	88	125	376	182	171	49	0.09	2.72	13.91	14.80	17.61
31	112	165	1394	354	965	127	0.17	5.50	82.13	112.91	118.58
32	105	141	210	81	114	39	0.17	5.49	15.97	12.78	18.43
33	91	125	71	41	24	21	0.17	4.51	8.00	4.08	8.77
34	624	855	6435	1479	4725	359	0.17	6.14	464.70	600.04	606.36
35	114	151	9	8	1	15	0.17	5.64	5.96	0.44	6.25
36	108	152	631	239	367	52	0.17	5.45	39.03	43.09	48.71
37	114	181	10	10	2	8	0.19	6.17	6.64	0.56	6.92
38	100	149	216	86	120	31	0.19	5.10	15.48	12.70	17.98
39	118	167	536	215	304	38	0.17	5.95	33.73	35.56	41.69
40	386	613	4694	1555	3246	104	0.19	6.53	468.17	600.13	606.85
avg.	142.35	199.90	779.55	230.40	531.60	50.05	0.14	4.32	60.41	73.91	78.38

Table 6.12: Computational results for random instances with 20 lots (d)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
1	2	3	0.2	0.8	10870.00	10880.00	10900	10900
2	2	3	0.2	0.8	9578.33	9600.00	9600	9600
3	2	3	0.2	0.8	9338.75	9340.00	9360	9360
4	2	3	0.2	0.8	9381.67	9400.00	9400	9400
5	2	3	0.2	0.8	9170.00	9180.00	9180	9180
6	2	3	0.2	0.8	8920.00	8940.00	8940	8940
7	2	3	0.2	0.8	10890.00	10900.00	10920	10920
8	2	3	0.2	0.8	7815.00	7820.00	7840	7840
9	2	3	0.2	0.8	9813.33	9840.00	9860	9860
10	2	3	0.2	0.8	9416.67	9420.00	9440	9440
11	5	2	0.2	0.8	16577.14	16600.00	16600	16600
12	5	2	0.2	0.8	16630.00	16640.00	16640	16640
13	5	2	0.2	0.8	17710.00	17720.00	17720	17720
14	5	2	0.2	0.8	16580.00	16600.00	16600	16600
15	5	2	0.2	0.8	16102.50	16120.00	16120	16120
16	5	2	0.2	0.8	16640.00	16660.00	16660	16660
17	5	2	0.2	0.8	16780.00	16800.00	16800	16800
18	5	2	0.2	0.8	17320.00	17320.00	17340	17340
19	5	2	0.2	0.8	17196.67	17200.00	17220	17220
20	5	2	0.2	0.8	16205.00	16220.00	16240	16240
avg.					13146.75	13160.00	13169.00	13169.00

Table 6.13: Computational results for random instances with 25 lots (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
1	20	25	23	15	8	3	0.03	0.20	0.44	0.25	0.48
2	69	74	97	24	45	45	0.02	0.72	1.83	1.42	2.16
3	47	64	5	2	2	2	0.02	0.53	0.55	0.06	0.61
4	49	59	21	7	8	10	0.03	0.59	0.75	0.28	0.91
5	48	59	44	22	17	14	0.03	0.59	1.03	0.59	1.22
6	52	58	12	5	6	14	0.02	0.63	0.72	0.14	0.78
7	31	35	68	28	36	20	0.03	0.31	1.02	0.80	1.14
8	54	78	272	106	152	45	0.02	0.70	3.84	4.74	5.46
9	51	56	510	68	390	71	0.03	0.56	5.35	7.15	7.74
10	45	52	63	29	26	18	0.02	0.47	1.15	0.89	1.37
11	118	156	20	5	9	12	0.13	4.61	5.29	0.89	5.62
12	103	126	49	33	15	15	0.13	3.79	5.40	2.03	5.95
13	95	115	260	129	128	22	0.11	3.52	13.11	11.74	15.37
14	131	167	19	0	9	20	0.11	5.20	5.85	0.89	6.20
15	123	185	19	12	8	10	0.13	5.45	6.16	0.95	6.53
16	113	147	80	52	29	14	0.13	4.64	7.91	3.82	8.58
17	112	160	166	60	90	33	0.11	4.86	11.73	8.58	13.55
18	139	171	39	26	18	9	0.13	5.81	7.40	1.91	7.84
19	132	162	25	17	8	8	0.13	5.77	6.66	1.16	7.05
20	96	138	240	151	80	28	0.11	4.17	14.41	12.00	16.28
avg.	81.40	104.35	101.60	39.55	54.20	20.65	0.07	2.66	5.03	3.01	5.74

Table 6.14: Computational results for random instances with 25 lots (b)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
21	6	2	0.2	0.8	18993.33	19000.00	19020	19020
22	6	2	0.2	0.8	20087.50	20100.00	20100	20100
23	6	2	0.2	0.8	18300.00	18300.00	18320	18320
24	6	2	0.2	0.8	18660.00	18680.00	18680	18680
25	6	2	0.2	0.8	19398.33	19420.00	19420	19420
26	6	2	0.2	0.8	18670.00	18680.00	18700	18700
27	6	2	0.2	0.8	20060.00	20080.00	20080	20080
28	6	2	0.2	0.8	18776.67	18800.00	18820	18820
29	6	2	0.2	0.8	21050.00	21100.00	21100	21100
30	6	2	0.2	0.8	18950.00	18960.00	18980	18980
31	8	2	0.2	0.8	24732.50	24740.00	24760	24800
32	8	2	0.2	0.8	25866.67	25900.00	25900	25900
33	8	2	0.2	0.8	24420.00	24440.00	24460	24460
34	8	2	0.2	0.8	25390.00	25400.00	25420	25420
35	8	2	0.2	0.8	24220.00	24240.00	24240	24260
36	8	2	0.2	0.8	25470.00	25480.00	25480	25480
37	8	2	0.2	0.8	24790.00	24800.00	24820	24820
38	8	2	0.2	0.8	25455.00	25460.00	25480	25480
39	8	2	0.2	0.8	24240.00	24260.00	24280	24280
40	8	2	0.2	0.8	26576.67	26600.00	26600	26600
avg.					22205.33	22222.00	22233.00	22236.00

Table 6.15: Computational results for random instances with 25 lots (c)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
21	113	155	130	59	62	23	0.17	7.07	14.83	8.92	16.16
22	146	192	28	13	13	13	0.17	8.46	9.93	1.81	10.44
23	101	140	19	16	2	6	0.16	6.84	8.07	1.36	8.36
24	132	175	88	45	35	21	0.17	7.83	13.11	6.36	14.36
25	129	178	27	8	10	15	0.17	8.16	9.75	2.03	10.36
26	111	164	28	11	16	14	0.17	7.28	8.99	2.10	9.55
27	150	194	5	0	5	4	0.17	9.00	9.22	0.41	9.58
28	132	189	59	26	28	14	0.17	8.22	11.79	4.16	12.55
29	138	171	67	38	25	10	0.19	8.01	11.54	4.32	12.52
30	106	151	65	45	24	12	0.17	6.23	10.06	4.31	10.72
31	149	209	2688	457	1548	239	0.28	15.09	469.46	600.31	615.68
32	133	195	8	0	4	13	0.28	13.83	14.37	0.88	14.98
33	142	195	1618	236	1317	88	0.28	14.82	268.26	301.12	316.21
34	147	192	61	43	18	15	0.30	13.86	20.44	7.31	21.47
35	168	241	2397	437	1524	258	0.27	16.89	498.07	600.61	617.77
36	147	197	77	42	32	22	0.28	13.78	21.68	8.82	22.88
37	140	203	45	24	19	13	0.30	15.36	20.99	6.30	21.96
38	151	189	1125	214	800	113	0.30	14.51	161.49	178.96	193.77
39	145	200	28	27	6	11	0.28	14.44	17.48	3.30	18.02
40	149	197	43	21	16	15	0.30	13.34	17.23	4.64	18.27
avg.	136.45	186.35	430.30	88.10	275.20	45.95	0.23	11.15	80.84	87.40	98.78

Table 6.16: Computational results for random instances with 25 lots (d)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
1	2	3	0.2	0.8	10897.41	10920.00	10920	10920
2	2	3	0.2	0.8	9940.42	9960.00	9960	9960
3	2	3	0.2	0.8	11896.67	11900.00	11920	11920
4	2	3	0.2	0.8	11245.00	11260.00	11260	11260
5	2	3	0.2	0.8	11167.78	11180.00	11200	11200
6	2	3	0.2	0.8	10534.17	10540.00	10540	10540
7	2	3	0.2	0.8	10503.33	10520.00	10520	10520
8	2	3	0.2	0.8	11703.33	11720.00	11720	11720
9	2	3	0.2	0.8	11393.33	11400.00	11400	11400
10	2	3	0.2	0.8	11465.00	11480.00	11480	11480
11	5	2	0.2	0.8	17420.00	17440.00	17440	17440
12	5	2	0.2	0.8	20433.33	20440.00	20460	20460
13	5	2	0.2	0.8	20960.00	20960.00	20980	20980
14	5	2	0.2	0.8	21077.50	21080.00	21080	21080
15	5	2	0.2	0.8	17810.00	17840.00	17860	17860
16	5	2	0.2	0.8	19383.33	19400.00	19400	19440
17	5	2	0.2	0.8	19940.00	19940.00	19960	19960
18	5	2	0.2	0.8	19780.00	19800.00	19800	19800
19	5	2	0.2	0.8	19586.67	19600.00	19600	19600
20	5	2	0.2	0.8	19010.00	19040.00	19040	19040
avg.					15307.36	15321.00	15327.00	15329.00

Table 6.17: Computational results for random instances with 30 lots (a)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
1	66	99	27	13	10	13	0.03	1.39	1.96	0.77	2.19
2	76	121	116	93	46	17	0.03	1.80	4.06	3.08	4.91
3	38	65	71	25	50	10	0.03	0.73	1.92	1.42	2.18
4	38	65	73	43	40	23	0.03	0.69	1.97	1.64	2.36
5	28	50	22	8	12	4	0.03	0.51	0.87	0.44	0.98
6	46	71	366	161	207	52	0.03	0.95	7.70	9.60	10.59
7	44	68	38	30	18	16	0.03	0.89	1.58	0.86	1.78
8	41	67	412	91	311	37	0.03	0.74	8.17	9.64	10.41
9	49	85	134	85	72	20	0.03	0.97	3.38	3.20	4.20
10	54	98	3	0	2	6	0.03	1.03	1.08	0.05	1.11
11	122	181	727	341	371	50	0.17	9.97	77.02	83.01	93.14
12	142	172	51	35	20	13	0.17	10.95	14.57	4.11	15.23
13	143	194	14	13	2	4	0.17	9.66	10.47	0.95	10.78
14	159	203	271	128	129	28	0.17	12.19	32.00	23.30	35.66
15	133	194	39	17	16	18	0.16	11.03	14.36	4.14	15.33
16	144	200	4753	425	3045	458	0.17	10.83	372.63	600.10	611.10
17	157	212	463	128	281	75	0.19	11.69	44.32	40.33	52.21
18	133	173	1980	426	1448	130	0.17	9.41	228.93	303.44	313.03
19	132	192	51	27	15	20	0.16	9.54	13.11	4.22	13.92
20	119	184	29	24	8	8	0.14	8.75	10.82	2.41	11.29
avg.	93.20	134.70	482.00	105.65	305.15	50.10	0.10	5.69	42.55	54.84	60.62

Table 6.18: Computational results for random instances with 30 lots (b)

	m	d	v_1	v_2	z_{LP}^{bc}	z_{LP}^{ac}	LB	UB
21	6	2	0.2	0.8	23400.00	23420.00	23440	23440
22	6	2	0.2	0.8	23183.33	23200.00	23200	23200
23	6	2	0.2	0.8	24875.00	24880.00	24900	24900
24	6	2	0.2	0.8	23561.25	23580.00	23580	23580
25	6	2	0.2	0.8	25467.14	25480.00	25480	25480
26	6	2	0.2	0.8	23323.33	23340.00	23340	23360
27	6	2	0.2	0.8	23703.33	23720.00	23720	23720
28	6	2	0.2	0.8	23260.00	23280.00	23280	23280
29	6	2	0.2	0.8	23800.00	23820.00	23820	23820
30	6	2	0.2	0.8	23926.67	23940.00	23940	23940
31	8	2	0.2	0.8	31105.00	31120.00	31120	31120
32	8	2	0.2	0.8	30144.29	30160.00	30160	30200
33	8	2	0.2	0.8	30598.33	30620.00	30620	30620
34	8	2	0.2	0.8	29285.00	29300.00	29300	29320
35	8	2	0.2	0.8	29727.22	29740.00	29740	29780
36	8	2	0.2	0.8	31202.50	31220.00	31220	31220
37	8	2	0.2	0.8	31850.00	31860.00	31880	31880
38	8	2	0.2	0.8	29865.00	29880.00	29880	29880
39	8	2	0.2	0.8	30865.00	30880.00	30880	30900
40	8	2	0.2	0.8	30390.00	30400.00	30420	30420
avg.					27176.62	27192.00	27196.00	27203.00

Table 6.19: Computational results for random instances with 30 lots (c)

	sp_{LP}	$cols_{LP}$	sp_{BB}	$cols_{BB}$	nod_{BB}	$cuts$	t_{PP}	t_{LP}	t_{SP}	t_{BB}	t_{TOT}
21	131	197	84	47	32	18	0.25	14.16	24.16	11.04	25.45
22	133	183	1895	266	1473	175	0.25	27.57	489.51	533.27	561.10
23	139	170	35	23	10	11	0.27	13.79	17.66	4.23	18.29
24	160	214	115	56	46	31	0.25	16.28	28.58	14.06	30.59
25	126	178	1960	362	1053	183	0.25	11.28	241.99	307.15	318.68
26	150	214	2108	501	1444	188	0.27	16.44	506.86	600.38	617.09
27	136	195	1978	419	1188	162	0.23	14.11	328.52	389.75	404.10
28	137	196	211	132	76	40	0.22	14.45	38.55	26.59	41.26
29	177	252	4	0	3	12	0.24	17.75	17.96	0.45	18.44
30	187	253	1513	258	852	148	0.23	19.55	279.51	316.43	336.22
31	172	230	785	293	461	69	0.41	28.05	231.45	229.21	257.67
32	181	244	1628	267	873	100	0.42	31.69	559.77	600.22	632.33
33	185	246	163	117	33	36	0.42	31.90	63.92	34.40	66.73
34	206	299	1685	304	1206	218	0.53	38.82	547.17	600.08	639.43
35	157	230	1505	301	728	228	0.42	28.94	566.64	602.26	631.62
36	150	207	11	5	5	10	0.45	26.16	28.05	2.31	28.92
37	164	228	116	70	34	30	0.44	27.18	48.91	23.90	51.51
38	179	261	365	188	156	47	0.44	31.73	109.68	87.13	119.30
39	189	247	1427	347	985	132	0.44	34.20	565.03	600.03	634.67
40	169	223	29	24	8	12	0.42	28.34	34.79	6.89	35.65
avg.	161.40	223.35	880.85	199.00	533.30	92.50	0.34	23.62	236.43	249.49	273.45

Table 6.20: Computational results for random instances with 30 lots (d)

6.9 Conclusion

In this chapter, we described a branch-and-price-and-cut algorithm for the Ordered Cutting Stock Problem. This is the first reported attempt to solve this problem exactly. The problem is particularly hard. It can be seen as a combination of the standard Cutting Stock Problem with the Traveling Salesman Problem. We formulate it with a column generation model that we solved using a branch-and-price-and-cut algorithm. Two families of valid cutting planes were used to strengthen the LP relaxations at each node of the branch-and-bound tree: subtour elimination constraints and comb inequalities. We used a branching scheme based on the arc flow model also described in this chapter. Our scheme is compatible with the pricing subproblem, as are the two types of cutting planes used. A simple rounding procedure was devised to accelerate the search for integer solutions.

A set of random instances was generated to test our approach. The instances had no more than 30 lots, since, for higher values, the capacity of our algorithm to find integer optima greatly deteriorates. Almost all the 200 instances were successfully solved within a time limit of 10 minutes of branch-and-bound.

Chapter 7

Conclusions

7.1 Contributions

Different variants of the one-dimensional cutting stock and bin-packing problems were addressed in this thesis. Their practical relevance is largely recognized in the field, and has been pointed out by many other authors. We investigated in particular problems in which more than a single type of large objects are available, problems with non-standard objectives, such as the minimization of the number of distinct patterns, and problems with new constraints on the ordering of the small items. We also studied methods to improve the convergence of the standard column generation algorithm. All the algorithms proposed were coded, and tested on many problem instances.

The results obtained throughout the thesis can be considered as state-of-the-art results. For the Multiple Length Cutting Stock Problem, and its packing counterpart the Variable Sized Bin-Packing Problem, the branch-and-price-and-cut algorithm studied in Chapter 3 produced better results than other algorithms described in the literature. In fact, only two alternative exact solution procedures have been reported so far. One of them consists in a combinatorial algorithm, while the other combines Chvatal-Gomory cutting planes with column generation. With our approach, we were able to solve all the instances the first author was unable to solve, and we get improved results with the instances of the second authors. For the Pattern Minimization Problem, we proposed to restrict the set of columns of the master problem so as to get a stronger linear relaxation. We proved that the cutting planes derived from dual feasible functions are always at least as strong as those used by Vanderbeck [123]. To get even stronger cuts, we proposed to derive them from surrogate constraints. We compared our approach to the one of Vanderbeck using the set of instances used by this author in [123]. Without any rounding heuristic, we solved more instances using less branching nodes. The algorithm developed for the Ordered Cutting Stock Problem is the first reported attempt to solve this problem to optimality. We were able to solve instances with up to 30 lots, and 8 different item sizes per lot. The bottleneck of the algorithm

is the pricing subproblem, whose resolution takes an important percentage of the total computing time.

The study of the stabilization technique introduced in [117] was extended to the whole branch-and-bound tree. We showed that, for the standard Cutting Stock Problem, some dual cutting planes may not be valid when used together with a specific branching scheme. Conditions were given, allowing one to select the dual cuts that are feasible in a node in which certain branching constraints were enforced. The computational results obtained with the application of these cuts in all the nodes of the branch-and-bound tree were very good. The computing times decreased substantially, as well as the number of branching nodes, and more instances could be solved to optimality.

The development of a branch-and-price algorithm implies a great part of customization. The branching schemes must be devised so as to be compatible with the pricing subproblem, keeping its original structure, or at least not complicating it too much. When cutting planes are necessary to strengthen models which are not strong enough to be tackled within a branch-and-bound framework, this development phase can become even more difficult. Not all types of cuts can be used a priori, at least without affecting the efficiency of the whole algorithm. Sometimes, good families of cuts have to be discarded, as happens with the cover inequalities, which can not be used for the Pattern Minimization Problem. All the branching schemes devised along this thesis ensure both convergence, and compatibility with the pricing subproblem. Moreover, cutting planes were chosen, or developed, with a similar objective in mind.

The way the algorithms were implemented in practice may have influenced the results presented in the thesis. We did not use any original paradigm for this codification, but we always tried to use the most efficient data structures, and to avoid bad practices in the development of our code. We did not include any reference to the details concerning this implementation. The main reason for doing so is because this is out the scope of this thesis, and also because the originality of our work is not there.

7.2 Future Research

A great part of the work that will be done in a near future is motivated by the weaknesses of our approaches, and by the horizons opened with the study of some of the subjects treated throughout this thesis. This work will be both practical, oriented to a more efficient resolution of practical problems in the field of cutting and packing, and theoretical, with an incidence in the area of integer programming methods such as column generation, branch-and-bound and cutting planes.

Given the capacity of dual cutting planes to accelerate column generation, and the results obtained when applying them in all the nodes of a branch-and-bound tree for

cutting and packing problems, it will be interesting to study how such an extension can be achieved for other problems, and with different branching schemes. Ideally, this study should be independent of the type of the problems, and should concentrate on general integer programs with a special structure.

The column generation formulation for the Pattern Minimization Problem remains quite weak, even after applying the cutting planes discussed in Chapter 5. To expect a more successful resolution of this problem with linear programming based branch-and-bound methods, an alternative model should be devised, or stronger cutting planes must be applied. For this purpose, we can follow the strategy used in this thesis, and investigate alternative dual feasible functions, or resort to other approaches for deriving valid cutting planes.

The dual feasible functions used for the Pattern Minimization Problem can be improved so as to derive stronger cutting planes for knapsack polytopes. An example was given in Chapter 5 illustrating how this can be done. As we said in that chapter, these improvements are easily applied with constraints whose coefficients are all known a priori. The way these new dual feasible functions can be used with a restricted master problem, and the quality of the cutting planes that can be derived, are the real questions. When the coefficients are dynamically generated, we must be able to anticipate the set of possible coefficients. It would be interesting to know the classes of cutting and packing problems for which this can be done, and if these problems have any practical relevance.

Regarding the Ordered Cutting Stock Problem, we are restricted to problems with up to 30 lots. An obvious challenge will be to tackle larger problems. Given its inherent complexity, this problem is an ideal testing ground for new solution approaches. In this domain, we can point to the actual trend that consists in mixing intensively branch-and-price-and-cut algorithms with heuristic approaches, including metaheuristics. The different decisions that are taken during the execution of a branch-and-price-and-cut algorithm can also be oriented heuristically. As an example, we refer to the way the nodes are selected, which is usually done in a standard style, using a depth-first search, or a best-bound search, among few others. It will be interesting to study how this can be done in practice, and if the use of more sophisticated approaches can have a real impact on the efficiency of branch-and-price-and-cut algorithms.

Bibliography

[1] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] R. K. Ahuja and J. B. Orlin. A fast and simple algorithm for the maximum flow problem. *Operations Research*, 37:748–759, 1989.

[3] J. M. Allwood and C. N. Goulimis. Reducing the number of patterns in one-dimensional cutting stock problems. Technical report, Electrical Engineering Department, Imperial College, London, 1988.

[4] C. Alves and J. M. Valério de Carvalho. Planeamento de rotas num sistema de recolha de desperdícios de madeira. *Investigação Operacional*, 24:21–43, 2004.

[5] L. H. Appelgren. A column generation algorithm for a ship scheduling problem. *Transportation Science*, 3:53–68, 1969.

[6] L. H. Appelgren. Integer programming methods for a vessel scheduling problem. *Transportation Science*, 5:64–78, 1971.

[7] B. M. Baker. A spreadsheet modelling approach to the assortment problem. *European Journal of Operational Reseaech*, 114:83–92, 1999.

[8] C. Barnhart, C. A. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity network flow problems. *Operations Research*, 48(3):318–326, 2000.

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

[10] M. S. Bazaraa, J. J. Jarvis, and H. D. Sherali. *Linear Programming and Network Flows*. John Wiley, New York, 2nd edition, 1980.

[11] J. E. Beasley. OR-library: Distributing test problems by electronic email. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[12] G. Belov. *Problems, Models and Algorithms in One- and Two- Dimensional Cutting*. PhD thesis, Dresden University, 2003.

[13] G. Belov and G. Scheithauer. A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths. *European Journal of Operational Research*, 141:274–294, 2002.

[14] H. Ben Amor. *Stabilisation de L'algorithme de Génération de Colonnes*. PhD thesis, École Polytechnique de Montréal, 2002.

[15] H. Ben Amor, J. Desrosiers, and J. M. Valério de Carvalho. Dual-optimal inequalities for stabilized column generation. *Les Cahiers du GERAD, G-2003-20*, 2003.

[16] E. E. Bischoff and M. S. W. Ratcliff. Loading multiple pallets. *Journal of the Operational Research Society*, 46:1322–1336, 1995.

[17] R. E. Bixby, J. W. Gregory, I. J. Lustig, R. E. Marsten, and D. F. Shanno. Very large-scale linear programming: A case study in combining interior point and simplex methods. *Operations Research*, 40(5):885–897, 1992.

[18] M. A. Carravilla, C. Ribeiro, and J. F. Oliveira. Solving nesting problems with non-convex polygons by constraint logic programming. *International Transactions in Operational Research*, 10:651–664, 2003.

[19] C.-L. S. Chen, S. M. Hart, and W. M. Tham. A simulated annealing heuristic for the one-dimensional cutting stock problem. *European Journal of Operational Research*, 93:522–535, 1996.

[20] C. Chu and R. La. Variable-sized bin packing: Tight absolute worst case performance ratios for four approximation algorithms. *SIAM Journal on Computing*, 30(6):2069–2083, 2001.

[21] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York, 1983.

[22] E. G. Coffman, M. R. Garey, and D. S. Johnson. Bin packing with divisible item sizes. *Journal of Complexity*, 3:406–428, 1987.

[23] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*. PWS Publishing Company, Boston, 1997.

[24] J. Csirik. An online algorithm for variable sized bin packing. *Acta Informatica*, 26(8):697–709, 1989.

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

[26] Z. Degraeve and M. Peeters. Optimal integer solutions to industrial cutting-stock problems: Part 2, benchmark results. *INFORMS Journal on Computing*, 15(1): 58–81, 2003.

[27] Z. Degraeve and L. Schrage. Optimal integer solutions to industrial cutting stock problems. *INFORMS Journal on Computing*, 11(4):406–419, 1999.

[28] G. Desaulniers, J. Desrosiers, I. Ioachim, M. Salomon, F. Soumis, and D. Villeneuve. A unified framework for deterministic time constrained routing and crew scheduling problems. In T. G. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, pages 57–93. Kluwer Academic Publishers, Boston, 1998.

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

[30] M. Desrochers and F. Soumis. A column generation approach to urban transit crew scheduling. *Transportation Science*, 23:1–13, 1989.

[31] J. Desrosiers, Y. Dumas, M. M. Solomon, and F. Soumis. Time constrained routing and scheduling. *Network Routing*, 8:35–139, 1995.

[32] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

[33] A. Diegel, M. Chetty, S. van Schalkwyk, and S. Naidoo. Setup combining in the trim loss problem: 3-to-2 2-to-1. Technical report, University of Natal, 1994.

[34] O. du Merle, J. Goffin, and J. Vial. On improvements to the analytic center cutting plane method. *Computational Optimization and Applications*, 11:37–52, 1998.

[35] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[36] H. Dyckhoff. A new linear programming approach to the cutting stock problem. *Operations Research*, 29:1092–1104, 1981.

[37] H. Dyckhoff. A typology of cutting and packing problems. *European Journal of Operational Research*, 44:145–159, 1990.

[38] H. Dyckhoff and U. Finke. *Cutting and Packing in Production and Distribution: A Typology and Bibliography.* Physica-Verlag, Heidelberg, 1992.

[39] E. Falkenauer. Tapping the full power of genetic algorithm through suitable representation and local optimization. In J. Biethann and V. Nissen, editors, *Evolutionary Algorithms in Management Applications*. Springer-Verlag, Berlin, 1995.

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

[41] A. A. Farley. A note on bounding a class of linear programming problems, including cutting stock problems. *Operations Research*, 38(5):922–923, 1990.

[42] S. Fekete and J. Schepers. New classes of fast lower bounds for bin packing problems. *Mathematical Programming*, 91:11–31, 2001.

[43] H. Foerster and G. Waescher. Simulated annealing for order spread minimization in sequencing cutting patterns. *European Journal of Operational Research*, 110 (2):272–281, 1998.

[44] H. Foerster and G. Waescher. Pattern reduction in one-dimensional cutting stock problems. *International Journal of Production Research*, 38(7):1657–1676, 2000.

[45] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[46] L. R. Ford and D. R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5:97–101, 1958.

[47] J. J. Forrest and D. Goldfarb. Steepest-edge simplex algorithms for linear programming. *Mathematical Programming*, 57:341–374, 1992.

[48] D. Friesen and M. Langston. Variable sized bin packing. *SIAM Journal on Computing*, 15:222–230, 1986.

[49] M. Garey and D. S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[50] T. Gau and G. Waescher. CUTGEN1: A problem generator for the standard one-dimensional cutting stock problem. *European Journal of Operational Research*, 84:572–579, 1995.

[51] A. M. Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Studies*, 2:82–114, 1974.

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

[53] P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting stock problem: Part II. *Operations Research*, 11:863–888, 1963.

[54] P. C. Gilmore and R. E. Gomory. Multistage cutting stock problems of two and more dimensions. *Operations Research*, 13:94–120, 1965.

[55] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum flow problem. *Journal of ACM*, 35:921–940, 1988.

[56] A. M. Gomes and J. F. Oliveira. A 2-exchange heuristic for nesting problems. *European Journal of Operational Research*, 14:359–370, 2002.

[57] C. Goulimis. Optimal solutions for the cutting stock problem. *European Journal of Operational Research*, 44:197–208, 1990.

[58] M. Gradisar, J. Jesenko, and G. Resinovic. Optimization of roll cutting in clothing industry. *Computers and Operations Research*, 24:945–953, 1997.

[59] M. Gradisar, M. Kljajic, G. Resinovic, and J. Jesenko. A sequential heuristic procedure for one-dimensional cutting. *European Journal of Operational Research*, 114:557–568, 1999.

[60] M. Gradisar, G. Resinovic, and M. Kljajic. Evaluation of algorithms for one-dimensional cutting. *Computers and Operations Research*, 29:1207–1220, 2002.

[61] R. W. Haessler. A heuristic programming solution to a nonlinear cutting stock problem. *Management Science*, 17(12):793–802, 1971.

[62] R. W. Haessler. Controlling cutting pattern changes in one-dimensional trim problems. *Operations Research*, 23(3):483–493, 1975.

[63] R. W. Haessler and P. Sweeney. Cutting stock problems and solution procedures. *European Journal of Operational Research*, 54:141–150, 1991.

[64] A. I. Hinxman. The trim-loss and assortment problems: A survey. *European Journal of Operational Research*, 5:8–18, 1980.

[65] J. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. A Series of Comprehensive Studies in Mathematics. Springer-Verlag, 1993.

[66] K. L. Hoffman and M. Padberg. Solving airline crew scheduling problems by branch-and-cut. *Management Science*, 39(6):657–682, 1993.

[67] O. Holthaus. Decomposition approaches for solving the integer one-dimensional cutting stock problem with different types of standard lengths. *European Journal of Operational Research*, 141:295–312, 2002.

[68] O. Holthaus. On the best number of different standard lengths to stock for one-dimensional assortment problems. *International Journal of Production Economics*, 83:233–246, 2003.

[69] ILOG. *CPLEX 6.5 User's Manual*. 1999.

[70] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993.

[71] M. P. Johnson, C. Rennick, and E. Zak. One-dimensional cutting stock problem in just-in-time environment. *Pesquisa Operacional*, 19(1):145–158, 1999.

[72] R. E. Johnston. Rounding algorithms for cutting stock problems. *Asia Pacific Operational Research Journal*, 3:166–171, 1986.

[73] R. E. Johnston. Cutting patterns and cutter schedules. *Asia Pacific Operational Research Journal*, 4:3–14, 1987.

[74] B. Kallehauge, J. Larsen, and O. B. G. Madsen. Lagrangean duality applied on vehicle routing with time windows. Technical Report IMM-TR-2001-9, Technical University of Denmark, Denmark, 2001.

[75] J. Kang and S. Park. Algorithms for the variable sized bin packing problem. *European Journal of Operational Research*, 147:365–372, 2003.

[76] L. V. Kantorovich. Mathematical models of organising and planning production. *Management Science*, 6:366–422, 1960. Translated from the Russian original, 1939.

[77] J. E. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8(4):703–712, 1961.

[78] K. Kim and J. L. Nazareth. The decomposition principle and algorithms for linear programming. *Linear Algebra and its Applications*, 152:119–133, 1991.

[79] N. Kinnersley and M. Langston. Online variable-sized bin packing. *Discrete Applied Mathematics*, 22(2):143–148, 1988.

[80] L. S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, New York, 1970.

[81] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141:241–252, 2002.

[82] A. Loebel. *Optimal Vehicle Scheduling in Public Transit*. PhD thesis, Technische Universitaet Berlin, 1997.

[83] M. E. Luebbecke and J. Desrosiers. Selected topics in column generation. *Les Cahiers du GERAD, G-2002-64*, 2002.

[84] G. S. Lueker. Bin packing with items uniformly distributed over intervals [a,b]. In *Proceedings 24th Annual Symp. Found. Comp. Sci. (FOCS 1983)*, pages 289–297.

[85] O. Marcotte. The cutting stock poblem and integer rounding. *Mathematical Programming*, 33:82–92, 1985.

[86] O. Marcotte. An instance of the cutting stock problem for which the rounding property does not hold. *Operations Research Letters*, 4(5):239–243, 1986.

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

[88] S. Martello and P. Toth. *Knapsack Problems*. Wiley, New York, 1990.

[89] C. McDiarmid. Pattern minimisation in cutting stock problems. *Discrete Applied Mathematics*, 98:121–130, 1999.

[90] M. Monaci. *Algorithms for Packing and Scheduling Problems*. PhD thesis, Università di Bologna, 2002.

[91] F. Murgolo. An efficient approximation scheme for variable-sized bin packing. *SIAM Journal on Computing*, 16:149–161, 1987.

[92] G. L. Nemhauser and S. Park. A polyhedral approach to edge coloring. *Operations Research Letters*, 10:315–322, 1991.

[93] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 1988.

[94] D. Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 46(5):758–767, 1997.

[95] C. T. Ragsdale and C. W. Zobel. The ordered cutting stock problem. *Decision Sciences*, 35(1):83–100, 2004.

[96] D. Rogers, R. Plante, R. Wong, and J. Evans. Aggregation and disaggregation techniques and methodology in optimization. *Operations Research*, 39:553–582, 1991.

[97] M. Ronnqvist. A method for the cutting stock problem with different qualities. *European Journal of Operational Research*, 83:57–68, 1995.

[98] G. Roodman. Near optimal solutions to one-dimensional cutting stock problem. *Computers and Operations Research*, 13:713–719, 1986.

[99] M. W. P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[100] G. Scheithauer and J. Terno. The modified integer round-up property of the one-dimensional cutting stock problem. *European Journal of Operational Research*, 84:562–571, 1995.

[101] G. Scheithauer, J. Terno, A. Mueller, and G. Belov. Solving one-dimensional cutting stock problems exactly with a cutting plane algorithm. *Journal of the Operational Research Society*, 52:1390–1401, 2001.

[102] K. E. Schilling. The growth of m-constraint random knapsacks. *European Journal of Operational Research*, 46:109–112, 1990.

[103] J. E. Schoenfield. Fast, exact solution of open bin-packing problems without linear programming. Technical report, US Army Space and Missile Defense Command, Alabama, USA, 2002.

[104] A. Scholl, R. Klein, and C. Jurgens. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers and Operations Research*, 24(7):627–645, 1997.

[105] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1986.

[106] P. Schwerin and G. Waescher. A new lower bound for the bin-packing and its integration to MTP. *Pesquisa Operacional*, 19(2):111–129, 1999.

[107] S. Seiden. An optimal online algorithm for bounded space variable-sized bin packing. *SIAM Journal Discrete Mathematics*, 14(4):458–470, 2001.

[108] S. Seiden, R. Stee, and L. Epstein. New bounds for variable-sized online bin packing. *SIAM Journal on Computing*, 32(2):455–469, 2003.

[109] H. Stadtler. A comparison of two optimization procedures for 1- and $1\frac{1}{2}$-dimensional cutting stock problems. *OR Spektrum*, 10:97–111, 1988.

[110] H. Stadtler. A one-dimensional cutting stock problem in the aluminium industry and its solution. *European Journal of Operational Research*, 44:209–223, 1990.

[111] P. E. Sweeney and R. W. Haessler. One-dimensional cutting stock decisions for rolls with multiple quality grades. *European Journal of Operational Research*, 44: 224–231, 1990.

[112] J. Teghem, M. Pirlot, and C. Antoniadis. Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem. *J. Comput. Appl. Math.*, 64:91–102, 1995.

[113] S. Umetani, M. Yagiura, and T. Ibaraki. One-dimensional cutting stock problem to minimize the number of different patterns. *European Journal of Operational Research*, 146:388–402, 2003.

[114] J. M. Valério de Carvalho. Exact solution of cutting stock problems using column generation and branch-and-bound. *International Transactions in Operational Research*, 5:35–44, 1998.

[115] J. M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86:629–659, 1999.

[116] J. M. Valério de Carvalho. LP models for bin-packing and cutting stock problems. *European Journal of Operational Research*, 141(2):253–273, 2002.

[117] J. M. Valério de Carvalho. Using extra dual cuts to accelerate convergence in column generation. *INFORMS Journal on Computing (in press)*, 2005.

[118] J. M. van den Akker, C. A. J. Hurkens, and M. W. P. Savelsbergh. A time-indexed formulation for single machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[119] P. H. Vance. Branch-and-price algorithms for the one-dimensional cutting stock problem. *Computational Optimization and Applications*, 9(3):211–228, 1998.

[120] P. H. Vance, C. Barnhart, E. L. Johnson, and G. L. Nemhauser. Solving binary cutting stock problems by column generation and branch-and-bound. *Computational Optimization and Applications*, 3(2):111–130, 1994.

[121] F. Vanderbeck. *Decomposition and Column Generation for Integer Programs.* PhD thesis, Université Catholique de Louvain, 1994.

[122] F. Vanderbeck. Computational study of a column generation algorithm for bin packing and cutting stock problems. *Mathematical Programming*, 86(3):565–594, 1999.

[123] F. Vanderbeck. Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem. *Operations Research*, 48(6):915–926, 2000.

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

[125] F. Vanderbeck and L. A. Wolsey. An exact algorithm for IP column generation. *Operations Research Letters*, 19:151–159, 1996.

[126] D. Villeneuve, J. Desrosiers, M. Luebbecke, and F. Soumis. On compact formulations for integer programs solved by column generation. *Les Cahiers du GERAD, G-2003-6*, 2003.

[127] G. Waescher, H. Haußner, and H. Schumann. An improved topology of cutting and packing problems. Technical Report 24, Faculty of Economics and Management, Otto van Guericke University Magdeburg, 2004.

[128] W. E. Wilhelm. A technical review of column generation in integer programming. *Optimization and Engineering*, 2:159–200, 2001.

[129] B. J. Yuen. Heuristics for sequencing cutting patterns. *European Journal of Operational Research*, 55:183–190, 1991.

[130] G. Zhang. Worst-case analysis of the FFH algorithm for online variable-sized bin packing. *Computing*, 56(2):165–172, 1996.