Università degli Studi di Milano

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dipartimento di Tecnologie dell'Informazione

Dottorato di Ricerca in Informatica, XVIII Ciclo
Settore scientifico/disciplinare INF-01

Ph.D. Thesis:

Branch-and-price algorithms for partitioning problems

Candidate: Alberto Ceselli

Tutor: Prof. Giovanni Righini

Coordinator: Prof. Giovanni Degli Antoni

A.A. 2004/2005 - November 2005

Acknowledgments

I firstly wish to thank my supervisor Giovanni Righini: in the last years he was consistently supporting and encouraging my research, always keeping me 'on the right track'.

Thanks to my colleagues Matteo, Nicola, Sandro and Luca, who shared with me good and bad times at the OptLab in Crema, and Andrea, who chose a different way.

I will always feel in debt to Annarita Bolzoni, who first seeded my interest in mathematics and computer science.

I thank, Sandro Fornili and Emanuele Angeleri, because during each discussion with them I catched glimpses of Science in the broad sense. A thank to the members of the Security Lab Group at the DTI, for the healthy collaboration on the minor research topic of my PhD.

I will always bear in mind the scientific and human example of Nelson Maculan. I am grateful to Rolf Möhring and the faculty of the Marie Curie training site in Berlin for welcoming me and giving me full support in completing my PhD, to Marco Lübbecke for the insightful discussions on many topics related to this thesis, and to the staff of the COGA group at the Technische Universität, for the (not only scientific) great time spent in Berlin. I wish to thank Pasquale Avella who readily agreed to review this thesis.

Federica, my parents and my brothers silently carried out all my moods in these years: without their support I would never have completed this work.

Finally, I wish to thank my 'friends in music' from the Made in Mind band and the Terzo Suono vocal group, for reminding me that there is still a world outside.

My PhD scholarship has been granted by the Italian MIUR (Ministero dell'Istruzione, dell'Università e della Ricerca); I also acknowledge the kind support of ACSU (Associazione Cremasca Studi Universitari) to the OptLab.

Contents

Preface

The need of understanding, modelling and controlling complex phenomena is becoming tighter and tighter in modern organized structures. A planner has often to make critical choices, involving large amounts of resource such as capitals: it is easy to imagine logistic or financial scenarios, in which managers have to provide long term investment plannings, design distribution systems for goods or ensure continuity in a supply chain.

Strategic planning is even more critical in location-allocation problems, in which the decision maker has to plan the distribution of facilities in a region. A typical example is the location of network servers, DSL concentrators or WLAN access points ensuring connectivity to telecommunication networks: in public context, where the target of the supplied service is often people, the possibility of providing a fair, quick and cheap access to resources, ensuring the *optimality* of the given planning, has an economic as well as a social and a political value. Furthermore, these considerations neglect extreme circumstances, like the location of emergency services or the design of protection systems for military targets. When such threats must be taken into account, each decision must be supported with quantitative and objective measures.

In all these situations, the simple experience is not sufficient to guarantee a deep understanding of the scenario and a conscious choice. This is why the support of special purpose information systems is needed. The development of such systems should consider two aspects, often antithetical: flexibility and efficiency.

Flexibility refers to the modeling aspect of the problem: a decision maker has to master powerful instruments for describing in an effective, quick and (first of all) rigorous way the observed system. He needs versatile tools, that could make easy to create, to correct and to consider variations of the built model. Mathematical programming, and in particular integer linear programming, gives the most appropriate theoretical background for realizing this kind of instruments.

On the other hand, efficiency refers to the algorithmic issues. In fact, in the general case, many kind of problems like the ones described before are shown to be "difficult" from a theoretical point of view. Nevertheless, theoretic complexity should not be a stumbling block for designing and implementing effective algorithms for solving these problems. The techniques devised in the combinatorial optimization scientific community perfectly integrate in this context.

The aim of this thesis is to move a step towards the achievement of a computational framework that combines flexibility with efficiency. We focus on the solution of *partitioning* problems. Our approach is the following: we consider mathematical models widely addressed in the literature, or generalizations of them, and we try to provide an effective algorithm for each of them, that is based on branch-and-price.

In fact, branch-and-price is an emerging method for solving hard combinatorial problems, which is based on implicit enumeration and column generation techniques.

The first contribution is therefore methodological: both applied mathematics and computer science are involved in the design of effective algorithmic techniques for this class of problems. The second contribution is application-oriented, since the tools obtained by implementing our methods are used for extensive experimentations on partitioning models. This yields an insight into the single problems as well as the possibility of comparing different models with similar tools.

The first chapter of the thesis introduces the whole work, addressing the class of partitioning problems and presenting the main theoretical background. It is not meant to be a comprehensive review of the argument, but rather a brief survey to focus the main topics of our research. Each of the subsequent chapters is a self-contained contribution.

In Chapters 3 and 4 we describe branch-and-price algorithms for two packing problems: the first one is a two-dimensional level strip packing problem and the second one is the ordered open-end bin packing problem. Both of them can be considered partitioning problems in which no cost is associated to the insertion of each element in a class; instead a fixed cost is associated to the activation of a class. We devote the whole Chapter 2 to describe our approach to the so-called penalized knapsack problem, that appears as a subproblem in these algorithms. In fact, our technique for the penalized knapsack problem showed to be a key issue in the design of effective procedures for this kind of packing problems.

Chapter 5 describes a branch-and-price algorithm for an extension of the generalized assignment problem, arising in production planning. This can be considered a representative of a problem of partitioning a set in a fixed number of classes, trying to minimize the average cost for inserting an element in a class.

In Chapters 6 and 7 we present a class of branch-and-price algorithms for single-source location problems, which combines characteristics of both assignment and packing problems. We start by describing our solution approach to the capacitated p-median problem. Then we extend it, devising a general purpose solver for location problems based on branch-and-price. Finally, this tool is used in an experimental comparison of several facility location models. Besides its aforementioned modelling and application-oriented contribution, this evaluation gives a final assessment of the effectiveness and flexibility of branch-and-price algorithms.

In Chapter 8 we briefly draw some conclusions and highlight further research directions.

Chapter 2 is based on the paper [19], earlier versions of the work in Chapters 3 and 4 were presented as [13] and [20] respectively, Chapter 6 contains the results published in [18], Chapters 5 and 7 appeared as technical reports [17] and [16] and are now submitted for publication.

Chapter 1

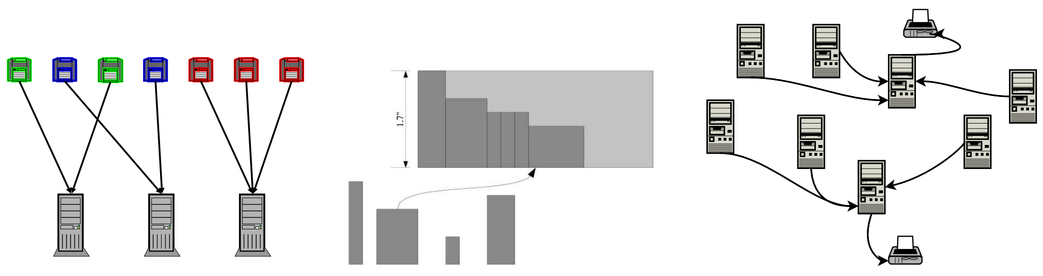
Theoretical background

In this chapter we briefly review the main mathematical tools used in the thesis. Since the main contribution of this work is given by the single applications, we felt that an homogeneous treatment of the subject lays outside our scope. Therefore, we refer to [100] and to the recent works of [27] and [26] for a detailed treatment of the subject. Indeed, in the aim of making the thesis self-contained, we tried to include in this chapter the theoretical notions used in the thesis.

1.1 Partitioning Problems

In a partitioning problem a set I of N elements must be assigned to a set J of M classes. This definition leads to a wide area of problems, ranging from statistical data clustering to network engineering and logistics. In this thesis we consider *optimization* problems, in which each class has a cost that depends on the elements composing the class, and the feasible classes can be described in terms of linear constraints.

Although these may seem very strong restrictions, several well-known models fall in this area, that are widely used as prototypes in real-world applications. The most famous one is probably the Generalized Assignment Problem (GAP) (Figure 1.1(a)). A set of agents must perform a set of tasks. Each agent has a finite amount of resource; in the minimization form of the problem, accomplishing each task implies time and resource consumption. The problem consists in finding the assignment of tasks to agents, in such a way that each agent has enough resources to complete its tasks and the total computing time is minimized. In this case, the assignment of tasks to agents is equivalent to partitioning the set of tasks in classes, one for each agent, and a single linear constraint suffices to impose that the resource requirement of the tasks in the same class do not exceed the resource available to the agent. In Chapter 5 we propose an algorithm for solving an



(a) (b) (c)

Figure 1.1: Partitioning problems: assignment (a), packing (b) and location (c)

extension of the GAP, in which each agent can perform tasks at different efficiency levels, trading resource consumption for time.

Partitioning is a powerful modeling tool also in logistics: in Chapters 3 and 4 we consider two variations of the strip packing and bin packing problem respectively (Figure 1.1(b)). In this case, a set of objects must be placed into containers. Clearly, the set of objects in each container forms a class of the partition. Geometric characteristics of both objects and containers, such as height and/or width, restrict the ways in which each class can be formed. The cost of the partition is given, in this case, by either the number of classes or a fixed cost for not leaving a class empty.

A third and wider case of study in this thesis is the class of single source location problems (Figure 1.1(c)): a set of customers, described by their position in a distribution network, has to be partitioned into clusters, and a facility must be activated in each of these clusters. In order to model real situations, often demands for customers and capacities for facilities must be considered. We studied location problems in which the demand of a customer has to be satisfied by exactly one facility. In Chapter 6 we present our algorithm for the Capacitated P-Median Problem (CPMP), while in Chapter 7 we tackle the problem of designing and implementing a tool for computer-aided location analysis.

1.2 Formulations and bounds

All the problems described above have already been addressed in the literature. Mathematical programming, and especially *integer linear programming*, showed to be the most appropriate modeling tool for this kind of problems. In particular, all of them admit a *compact formulation*, that is a formulation involving a polynomial number of both variables and constraints. This is of the following form:

$$\text{SPP)}\quad \min\ v = \sum_{j\in J} c^j(x^j)$$

$$\text{s.t.} \sum_{j\in J} x^j = 1 \tag{1.1}$$

$$A^j x^j \geq b^j \qquad \forall j \in J \tag{1.2}$$

$$x^j \in \{0,1\}^N \qquad \forall j \in J \tag{1.3}$$

Each class j in the partition is described by a binary vector x^j; each component x^j_i is 1 if element i belongs to class j, 0 otherwise. Constraints (1.1) are referred as *partitioning constraints.* Together with integrality conditions (1.3), they enforce that each element belongs to exactly one class, while each block j of constraints (1.2) describes how each class can be composed in a feasible partition. The cost for creating each class j is a function $c^j()$ of the elements in the class. The objective is to minimize the average cost of the classes in the partition.

The complexity of partitioning problems depends on the structure of the constraint matrix for each class ($A^j x^j \geq b^j$), but in the general case, as for the problems introduced in the previous section, partitioning problems are $\mathcal{NP}$-hard [41]. Hence, resorting to implicit enumeration algorithms to obtain a proven optimal solution is legitimate; this implies that a great effort has to be spent in the engineering of an effective method, also from an experimental point of view.

A first building block for these frameworks is a good estimate of the optimal solution. This is obtained by *relaxations* of the problem. The value of these relaxations is often called *dual* bound, since it is a feasible solution for a dual problem.

1.2.1 The linear programming (LP) relaxation

As long as the $c^j()$ functions are linear, or can be linearized, a valid dual bound can be obtained by relaxing the integrality conditions (1.3) and solving the corresponding LP.

$$\text{L-SPP)}\quad \min\ v = \sum_{j\in J} c^j(x^j)$$

$$\text{s.t.} \sum_{j\in J} x^j = 1 \tag{1.4}$$

$$A^j x^j \geq b^j \qquad \forall j \in J \tag{1.5}$$

$$0 \leq x^j \leq 1 \qquad \forall j \in J \tag{1.6}$$

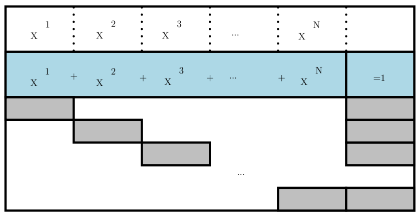


Figure 1.2: Structure of the constraint matrix in a partitioning problem

If the polyhedron associated to the whole formulation does not coincide with the convex hull of its integral points, the optimal solution of L-SPP may be fractional. Therefore, this LP bound is commonly strengthened by adding valid inequalities to the initial formulation [83], with the aim of approximating the convex hull. In many cases, this intent is pursued by considering only a subset of constraints, and describing the convex hull of the corresponding space. The set of constraints (1.2) is often a good candidate for deriving tight inequalities.

In branch-and-cut algorithms the generation of valid inequalities is dynamically performed in order to cut off fractional solutions, and coupled with the exploration of a search tree. Nowadays, branch-and-cut is a very well understood technique, which is implemented in several general-purpose optimization packages like CPLEX [1], X-Press [101] and GLPK [57]. It is effective in solving a wide range of mixed integer programs.

1.2.2 The Lagrangean relaxation

An alternative way of tackling the problem is that of *decomposition*. As depicted in figure 1.2, the constraint matrix of a partitioning problem has a particular structure: neglecting the block of partitioning constraints, this is a block diagonal matrix. Therefore, by dropping constraints (1.1) the problem decomposes into M independent (and smaller) subproblems. A better way of exploiting this property is to use Lagrangean relaxation [76], in which the violation of the relaxed constraints is penalized with suitable terms in the objective function:

$$\min \ v(\lambda) = \sum_{j \in J} c^j(x^j) - \lambda(\sum_{j \in J} x^j - 1)$$

$$A^j x^j \geq b^j \qquad \forall j \in J \qquad (1.7)$$

$$x^j \in \{0,1\}^N \qquad \forall j \in J \qquad (1.8)$$

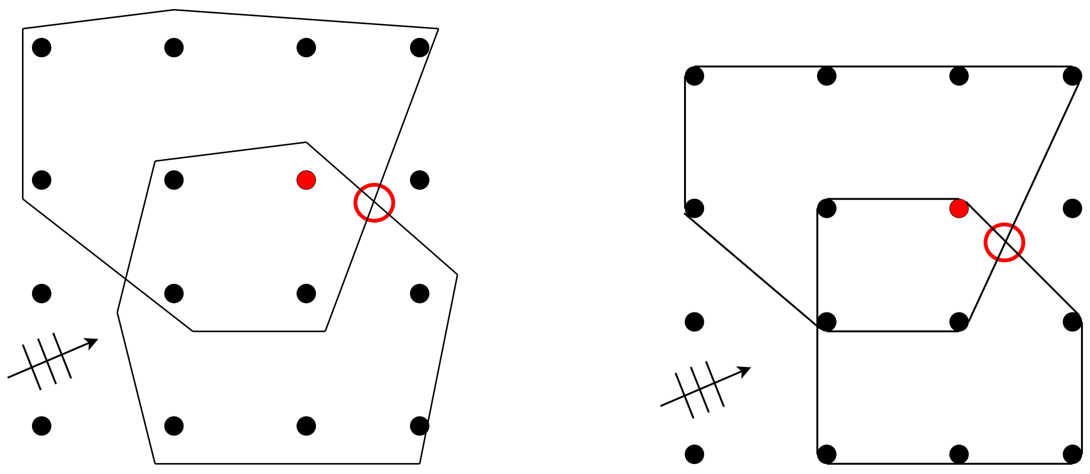


Figure 1.3: Convexification of two set of constraints

As described, the problem decomposes into M independent subproblems, each of the form:

$$\begin{aligned} \min \quad & v^j(\lambda) = c^j(x^j) - \lambda x^j \\ & A^j x^j \geq b^j \\ & x^j \in \{0,1\}^N \end{aligned}$$

The elements of the vector λ are usually indicated as *Lagrangean Multipliers*. For any choice of λ, the value obtained by adding the optimal solution values of all these subproblems is a valid dual bound for SPP; the problem of finding the vector λ which gives the tightest bound is called the *Lagrangean Dual* problem. Let each $\Omega^j = \{x^j | A^j x^j \geq b^j, 0 \leq x^j \leq 1\}$ be the region described by each block j of constraints (1.7) and (1.8). The following central result assesses the quality of the Lagrangean bound [42] [76]:

Proposition 1 *The bound obtained by solving the Lagrangean Dual to optimality is the same obtained by solving the LP relaxation of SPP, in which the M regions described by constraints $A^j x^j \geq b^j$ are substituted by their convex hulls (Figure 1.3).*

Therefore, if the vertices of the polyhedra Ω^j can have fractional coordinates, they are said not to have the *integrality property*, and the bound obtained in this way dominates the bound obtained by the LP relaxation; of course, for a particular (suboptimal) choice of the λ vector this relation may not be true.

Several methods have been devised to either solve the Lagrangean Dual to optimality, or approximating it. In the first class fall the bundle-like and analytic center methods [37]. They share a feature with column-generating algorithms: they iteratively solve a set of subproblems, in order to obtain a feasible relaxation, and a master problem, in order to update the multipliers vector. Their convergence properties are related to the complexity of the master problem. The subgradient-like algorithms fall in the second class of methods: they try to find a good vector of multipliers, trading accuracy and guarantees with speed. To this category belong also recent techniques like the Volume Algorithm [7] [4], in which the master problem of the bundle methods is treated in an aggregated form.

1.2.3 Dantzig-Wolfe decomposition

The approach of Dantzig and Wolfe yields the same result through a different way. First, consider the LP relaxation of SPP in the following form:

$$\begin{aligned} \min \quad & v = \sum_{j \in J} c^j(x^j) \\ \text{s.t.} & \sum_{j \in J} x^j = 1 \\ & x^j \in \Omega^j \qquad\qquad\qquad\qquad \forall j \in J \end{aligned}$$

The starting point for the Dantzig-Wolfe method is the following theorem [76]:

Theorem: Each point in Ω^j can be expressed as a convex combination of the extreme vertices $\bar{x}^j_k, k \in K^j$ and a linear combination of extreme rays $\bar{r}^j_l, l \in L^j$ of Ω^j.

That is, for each $x^j \in \Omega^j$,

$$x^j = \sum_{k \in K^j} \bar{x}^j_k z^j_k + \sum_{l \in L^j} \bar{r}^j_l h^j_l$$

with $\sum_{k \in K^j} z^j_k = 1$.

Furthermore, if the Ω^j polyhedra are bounded, each point can be described in terms of extreme vertices only. Since this is always our case, in the subsequent paragraphs we drop the indication about extreme rays.

The original formulation of SPP can be rewritten in the following form, called *master problem* (MP):

$$\begin{aligned}\min\ \ v = \sum_{j\in J} c^j (\sum_{k\in K^j} \bar{x}^j_k z^j_k)\\ \text{s.t.} \sum_{j\in J}(\sum_{k\in K^j} \bar{x}^j_k z^j_k) = 1\\ \sum_{k\in K^j} z^j_k = 1\end{aligned}$$

This new model has a variable for each extreme point of the Ω^j polyhedra. Their number can grow exponentially in the dimensions of the problem. However, this gives the following advantage: each polyhedron Ω^j can be replaced by its convex hull $conv(\Omega^j)$, by simply replacing the set $\{\bar{x}^j_k, k \in K^j\}$ of extreme points of Ω^j with the set $\{\bar{x}^j_k, k \in \bar{K}^j\}$ of the extreme points of $conv(\Omega^j)$.

$$\min\ \ v = \sum_{j\in J} c^j (\sum_{k\in \bar{K}^j} \bar{x}^j_k z^j_k)$$

$$\text{s.t.} \sum_{j\in J}(\sum_{k\in \bar{K}^j} \bar{x}^j_k z^j_k) = 1 \tag{1.9}$$

$$\sum_{k\in \bar{K}^j} z^j_k = 1 \tag{1.10}$$

Theoretically, the bound found by solving this LP and the one obtained by solving the Lagrangean Dual Problem to optimality are the same. However, this relaxation cannot be computed directly, since the number of variables involved in the optimization is too large. Therefore, practical implementations of this technique use column generation methods [43].

This technique generalizes the simplex method in the following way: as in the simplex algorithm, the first step is to find a feasible basis, that is a subset of as many variables as the number of constraints in the LP. The value of the variables outside the basis is fixed to 0, and the remaining problem is basically a linear system of equations, and can be solved efficiently. This yields an initial primal and a corresponding dual solution. Then, the simplex algorithm tries to improve the selected basis. This is done by looking at the reduced cost of the non-basic variables. If no non-basic variable has negative reduced cost (for the case of minimization problems), the current basis, and therefore the current solution, is optimal. Otherwise, a non-basic variable with negative reduced cost is included into the basis, a basic variable is excluded correspondingly, and the pricing process is iterated. From a geometric point of view, this corresponds to iteratively moving from an extreme vertex of the polyhedron associated to the LP to an adjacent extreme vertex, following the direction of the objective function.

As long as the value of a variable is fixed to 0, there is no need of storing the corresponding column. Hence, the column generation process starts by considering a restricted MP (RMP), containing a tractable subset of columns, that satisfies the only condition of containing a feasible basis. This RMP is optimized, obtaining a primal and a dual solution. Then, the pricing step is modified: instead of computing explicitly the reduced cost of each non-basic variable, an *optimization* problem is solved, whose aim is to find the variable with the lowest reduced cost. If the reduced cost of this variable is non-negative, the current basis is optimal, and therefore the solution of the RMP is optimal also for MP. Otherwise, the column corresponding to the new variable is inserted into the RMP, the problem is re-optimized and the pricing process is iterated. It is easy to notice that any such variable cannot not be in the RMP, since it would be included into the basis during the RMP optimization. From a geometric point of view, this corresponds in moving to the vertex associated to the optimal RMP solution with simplex iterations, that is the farthest known vertex of the polyhedron associated to MP, and to *generate* the nearest vertices by need (see Figure 8.1).

Let λ and μ be the vectors of dual variables associated to constraints (1.9) and (1.10) respectively. In our case, using a standard pricing rule, the reduced cost of any variable z_k^j is:

$$rc(z_k^j) = c^j(\bar{x}_k^j) - \lambda \bar{x}_k^j - \mu_j$$

and therefore, the problem of finding the vertex $\bar{x}_k^j$ corresponding to the variable with the lowest reduced cost is:

$$\min \ rc(z_k^j) = c^j(\bar{x}_k^j) - \lambda \bar{x}_k^j - \mu_j \tag{1.11}$$
$$\text{s.t.} \ \ A^j \bar{x}_k^j \geq b^j \tag{1.12}$$
$$\bar{x}_k^j \in \{0,1\}^N. \tag{1.13}$$

Neglecting the μ_j term, that is not involved in the optimization process, these are the same subproblems as in the Lagrangean relaxation.

Therefore, Dantzig-Wolfe decomposition with this column generation scheme can be viewed as a finite method for solving the Lagrangean Dual problem to optimality, as well as obtaining the corresponding primal solution. The optimality and finiteness of this method come at the price of explicitly storing (a subset of) the solutions of each Lagrangean relaxed problem. We remind that, in subgradient-like algorithms, the knowledge about past solutions is only approximated, and so it is the quality of primal and dual solutions.

On the other hand, the equivalence with Lagrangean relaxation can be used to improve column generation algorithms. In fact, the optimal value of the RMP relaxation is a valid dual bound for SPP only at the end of the column generation process (once optimality for MP is proved), while the value of the Lagrangean relaxation provides a valid dual bound at each column generation iteration.

The recurrent idea in the following chapters is to combine the good characteristics of these two methods in a unique framework.

Part I

Packing Problems

We start by considering packing problems. In this kind of partitioning problems, a set of items must be organized in bins. There are no allocation costs, but only fixed costs for activating a bin.

First, in Chapter 2 we study a penalized knapsack problem (PKP). This is a variation of the binary knapsack problem, arising as a subproblem in the subsequent algorithms. The main peculiarity of this variation is that the items are `ordered`*, and a penalty term is associated to the last selected item. This structure occurs in several well-known models, as well as in real applications. Although directly solving this problem with knapsack problem codes is impractical, we show how to exploit the vast knowledge on the topic to devise effective solution strategies.*

Then our algorithm is used as a pricing routine in a column generation algorithm for the two-dimensional level strip packing problem, discussed in Chapter 3. Besides assessing the overall effectiveness of branch-and-price against a general purpose solver for this kind of packing problems, we verified that actually our pricing technique makes the difference in designing effective methods.

Finally, in Chapter 4, we consider the ordered open-end bin-packing problem, which was previously tackled only from an approximation point of view [103]. Again, we tried to solve the problem with branch-and-price, and we exploited the special ordered structure of the items to devise efficient methods for the pricing routine. We also show how to embed information drawn from combinatorial properties of the problem in a branch-and-price framework.

Chapter 2

An optimization algorithm for a penalized knapsack problem

We study a penalized knapsack problem, that is a variation of the 0-1 knapsack problem, in which each item has a profit, a weight and a penalty. A subset of items has to be selected such that the sum of their weights does not exceed a given capacity and the objective function to be maximized is the sum of the profits of the selected items minus the largest penalty associated with the selected items. We present an integer linear programming formulation and we describe an exact optimization algorithm. The experimental analysis on a testbed of more than 3000 randomly generated instances of different size with different types of correlation between profits, weights and penalties shows that our algorithm is two orders of magnitude faster than a state-of-the-art general purpose solver.

2.1 Introduction

Penalized knapsack problems are variations of the 0-1 knapsack problem, in which a subset of items must be selected such that the sum of their weights does not exceed a given capacity and the overall value of the objective function to be maximized is given by the sum of the profits associated with the selected items minus a penalty term. An example of a penalized knapsack problem, in which the penalty is a function of the total capacity used has been presented in [38]. We study a penalized knapsack problem in which each item has a profit, a weight and a penalty. The objective function to be maximized is the sum of the profits of the selected items minus a penalty, which is the maximum among the penalties associated with the selected items.

Our motivation is similar to that of Freling et al. [38], since it is related to the pricing problem arising in a branch-and-price approach to solve the set covering reformulation of a difficult integer linear problem. In particular our penalized knapsack problem (PKP in the remainder) arises as a pricing subproblem in branch-and-price algorithms for the two-dimensional level strip packing problem (2LSPP) [67]. The 2LSPP consists of packing a set of rectangular items of given width and height into a strip of given width and variable height divided into levels, subject to the constraint that the items inserted in each level cannot be put on top of one another; hence the height of each level depends on the maximum height of the items inserted in it. The objective is to minimize the overall height of the strip. When the 2LSPP is reformulated as a set covering problem in which each column corresponds to a feasible subset of items inserted into the same level, the pricing subproblem is a PKP, in which the weight of each item is its width, the profit is the corresponding dual multiplier and the penalty is its height. This raises the need of fast optimization algorithms for the PKP.

In spite of the vast scientific literature on the 0-1 knapsack problem (see for instance [74] and [72]), we are not aware of any previous attempt to formulate and solve this PKP, either from the viewpoint of exact optimization or from that of heuristic approximation. In this paper we present an algorithm to solve the problem to proven optimality. We also analyze the effect of different degrees of correlation between weights, profits and penalties.

2.2 Formulation

A set $\mathcal{M}$ of N items is given; each item $j \in \mathcal{M}$ has a weight a_j, a profit c_j and a penalty p_j. A given capacity b is available. A mixed-integer linear programming formulation for the PKP is the following.

$$\max \quad z = \sum_{j \in \mathcal{M}} c_j x_j - \eta \tag{2.1}$$

$$\text{s.t.} \sum_{j \in \mathcal{M}} a_j x_j \le b \tag{2.2}$$

$$p_j x_j - \eta \le 0 \qquad \forall j \in \mathcal{M} \tag{2.3}$$

$$x_j \in \{0, 1\} \qquad \forall j \in \mathcal{M}$$

Each binary variable x_j takes value 1 if and only if item j is selected. Constraint (2.2) imposes that the sum of the weights of the selected items does not exceed the available capacity. The free variable η represents the penalty term: constraints (2.3) impose that η is greater than or equal to the penalty of any selected item.

Throughout the paper we make the assumptions that coefficients a_j are non-negative integers and coefficients c_j and p_j are non-negative. We also assume that $a_j \le b \ \forall j \in \mathcal{M}$ (i.e. each item can fit into the knapsack) and $\sum_{j \in \mathcal{M}} a_j > b$

(otherwise the capacity constraint would be redundant and the problem would be trivial). We also assume that the items have been ordered by non-increasing values of their penalty p_j, so that $j_1 < j_2$ implies $p_{j_1} \geq p_{j_2}$. In the remainder we define as *leading item* of a solution x the item j^* such that $x_{j^*} = 1$ and $p_{j^*} \geq p_j x_j \; \forall j \in \mathcal{M}$, that is the item with the maximum penalty among those which belong to the solution. Obviously $\eta = p_{j^*}$ at optimality.

The PKP is $\mathcal{NP}$-hard in the weak sense like the knapsack problem (KP). The KP is a special case of the PKP arising when $p_j = 0 \; \forall j \in \mathcal{M}$. If the leading item j^* of the optimal solution were known, the complete solution could be computed by discarding all items with $j < j^*$ and solving a KP considering the remaining items. Therefore a simple-minded algorithm for the PKP consists of iteratively fixing each of the N items as the leading item and to solve the corresponding KP. Since the KP can be solved in $O(Nb)$ time, this method yields a pseudo-polynomial algorithm with time complexity $O(N^2b)$ for the PKP.

2.3 An algorithm for the PKP

Once the leading item is selected, the PKP reduces to a KP, that can be solved by very effective algorithms [88] [72]. However instead of solving N different knapsack problem instances, one for each possible choice of the leading item, one would like to solve a smaller number of them. To this purpose our algorithm performs an exhaustive search to identify the optimal leading item.

General description. The algorithm sorts the items by non-increasing penalty and it initializes a best incumbent lower bound, z_{LB}, and a set of candidate leading items, S. Then the algorithm computes upper bounds to the value of the PKP for each possible choice of the leading item. These upper bounds are used both to guide the search in a best-first-search fashion and to terminate the algorithm. After that the algorithm iteratively selects a most promising leading item according to its associated upper bound, it solves a corresponding binary knapsack problem and this yields a feasible PKP solution. The information provided by the optimal solution of the binary knapsack problem is also exploited by additional fathoming rules and domination criteria to reduce the number of possible candidate leading items to be considered.

Preprocessing and initialization. The algorithm requires to sort the items by non-increasing value of p_j. For sorting the items we used a standard quicksort algorithm (implemented by the *qsort* library function of the ANSI C language), with median pivot selection [48]. Once the items have been sorted, we consider the range $\{l, \ldots, N\}$ such that $\sum_{j=l}^{N} a_j \leq b$ and $\sum_{j=l-1}^{N} a_j > b$. The optimal solution of the PKP involving only items in $\{l, \ldots, N\}$ can be computed in linear time since the capacity constraint is inactive. This optimal value is kept as an initial lower bound z_{LB} and all items in the range $\{l, \dots, N\}$ are no longer considered as candidate leading items.

Reduction. Some more items that cannot be optimal leading items are identified as follows. Whenever $c_j \leq p_j - p_{j+1}$, item j can be discarded from the set S of candidate leading items. Given a PKP solution with j as a leading item a non-worse PKP solution can be obtained by simply deleting item j from it, since the decrease in the penalty term η is not less than the loss in the sum of profits.

Notation. In the remainder we use the following notation. With $KP(j)$ we indicate the optimal value of the binary knapsack problem in which the only items available are those in the range $[j, \dots, N]$.

$$KP(j) = \max \ \{\sum_{i=j}^{N} c_i x_i : \sum_{i=j}^{N} a_i x_i \leq b, x_i \in \{0,1\} \ \forall i = j, \dots, N\}$$

With KP_j we indicate the optimal value of the binary knapsack problem in which item j is the leading item.

$$KP_j = c_j + \max \ \{\sum_{i=j+1}^{N} c_i x_i : \sum_{i=j+1}^{N} a_i x_i \leq b - a_j, x_i \in \{0,1\} \ \forall i = j+1, \dots, N\}$$

With PKP_j we indicate the optimal value of the penalized knapsack problem in which item j is the leading item. Hence we have:

$$PKP_j = KP_j - p_j \tag{2.4}$$

Finally we indicate by LKP_j the optimal value of the linear relaxation of the binary knapsack problem in which the leading item j has been fixed.

$$LKP_j = c_j + \max \ \{\sum_{i=j+1}^{N} c_i x_i : \sum_{i=j+1}^{N} a_i x_i \leq b - a_j, 0 \leq x_i \leq 1 \ \forall i = j+1, \dots, N\}$$

To describe the algorithm we use the following inequalities.

$$p_j \geq p_{j+1} \tag{2.5}$$

This is true because of the preliminary ordering of the items.

$$KP(j) \geq KP(j+1) \tag{2.6}$$

$$KP(j) \geq KP_j \tag{2.7}$$

$$LKP_j \geq KP_j \tag{2.8}$$

These three inequalities are true because the values on the left hand sides are optimal values of relaxations with respect to the optimal values on the right hand sides.

Step 1: computation of upper bounds. It has been observed that the optimal fractional value of the continuous knapsack problem is often a good estimate of the integer one [88]. This suggests that the most promising leading items can be identified from the values of the corresponding linear relaxations. Hence the first step of our algorithm consists of the computation of an upper bound μ_j for each possible choice of the leading item $j = 1, \ldots, N$.

$$\mu_j = LKP_j - p_j \quad \forall j = 1, \ldots, N \tag{2.9}$$

Proposition 2. *The value of μ_j is an upper bound to the optimal value of the PKP in which j is the leading item.*

$$\mu_j \geq PKP_j \quad \forall j = 1, \ldots, N \tag{2.10}$$

Proof. From definitions (2.4) and (2.9) and from inequality (2.8) it follows $\mu_j = LKP_j - p_j \geq KP_j - p_j = PKP_j$. $\Box$

The computation of each upper bound μ_j requires the optimization of a continuous knapsack problem, that can be carried out in $O(N)$ time [5]: hence the computation of all the upper bounds takes $O(N^2)$ time. However, instead of solving N continuous knapsack subproblems, the optimal solution of each of them can be obtained by suitably exploiting the structure of the optimal solution of the previous one and this yields a significant reduction in computing time. Consider the *efficiency* of each item j, that is the ratio $e_j = c_j/a_j$ and consider a sorting of the items by non-increasing value of efficiency. The optimal solution of a continuous knapsack problem can be found by selecting items according to the efficiency order, until an item is found whose weight exceeds the residual capacity. In order to fill the knapsack in an optimal way this item, called *break item*, is taken with a fractional value. In our algorithm we keep a pointer to the break item after each optimization; in the subsequent computation the previous leading item is removed from the knapsack and discarded, while a new leading item is inserted into the knapsack. This yields to either a violation or a slack in the capacity constraint. In the former case the new optimal solution is obtained from the previous one by removing items (or fractions of items) starting from the break item backward; in the latter case the new optimal solution is obtained by adding items (or fractions of items) from the break item forward. The worst-case computational complexity of this procedure is still $O(N^2)$; In fact, consider the following instance

$$p_j = \begin{cases} 1 & \text{if } j \text{ is odd} \\ 2+\epsilon & \text{if } j \text{ is even} \end{cases} \quad w_j = \begin{cases} n/2 & \text{if } j \text{ is odd} \\ 1 & \text{if } j \text{ is even} \end{cases} \quad h_j = n - j + 1 \quad c = n/2$$

where ϵ is any small positive constant. Only the item 1 is selected in the first LKP relaxation, and in the second LKP relaxation item 1 is removed and the $n/2$ items with even indices are added. Then, in the third LKP relaxation, item 2 is removed, item 3 is inserted and $n/2-1$ items with even indices removed. Hence, the total number of steps is $\sum_{j=1}^{n}(n-j+1)/2 = n/2 \cdot (n+1)/2 \in O(n^2)$.

Thus the complexity of the two initial sortings is dominated by the subsequent computations. However, we observed that on the average, a few iterations are needed for computing each μ_j value subsequent to the first. Notice that the test between brackets in the figure need not to be evaluated, as for each j we are fixing a set of variables to 0, and each weight is less than or equal to the capacity value. Therefore, a feasible LKP solution always exists.

Step 2: search. In the second step of our algorithm at each iteration the most promising leading item k is chosen: $k = \text{argmax}_{j\in S}\{\mu_j\}$ where S is the set of indices of candidate leading items not yet considered or fathomed. As soon as μ_k is found to be not greater than the best incumbent lower bound z_{LB}, the algorithm terminates. Once the most promising item k has been selected, a binary knapsack problem is solved, where the only available items are those with index not less than k.

To solve binary knapsack problem instances we used Pisinger's MINKNAP algorithm [88], that is very fast and exploits the optimal solution of the continuous relaxation both as a dual bound and to identify a good starting primal solution. MINKNAP dynamically expands the core of the knapsack problem and this can be well exploited in the PKP context: when we sort the items by efficiency, we break ties by rewarding items with higher penalty value. Since the bounds become tighter and tighter as the core is expanded, this choice reduces the probability of complementing variables with very high or very low penalty values. We observed a significant improvement in computation time when we applied this technique for solving subset-sum problems (see Section 2.4).

Every time we optimize a binary knapsack problem we get an optimal value $KP(k)$ but also a feasible solution to the PKP, which is obtained as follows. Let $\overline{x}$ be the optimal solution of the binary knapsack problem.

$$\overline{x} = \text{argmax}\{\sum_{i=k}^{N} c_i x_i : \sum_{i=k}^{N} a_i x_i \leq b, x_i \in \{0,1\}\ \forall i = k,\ldots,N\}$$

The structure of $\overline{x}$ allows to apply a dominance criterion. Let h be the leading item in $\overline{x}$, that is

$$h = \min\ \{i : \overline{x}_i = 1\}$$

Then the following proposition holds.

Computing the μ_j relaxation values

Input: the capacity value c and the set $\mathcal{M}$ of items j, each having a weight w_j, a price p_j and a weight w_j

Output: a set of μ_j values, representing the upper bound values

```
sort items $j \in \mathcal{M} = \{1,\dots,n\}$ by non-increasing value of efficiency $p_j/w_j$
for each $j \in \mathcal{M}$ do $s_j := 0.0$
$j := 1$
prev := null /* where $p_{null} = 0$, $w_{null} = 0$ and $\mu_{null} = 0$ */

while $\mathcal{M} \neq \emptyset$ do
    $j^* := \text{argmax}_{j \in \mathcal{M}}\{h_j\}$
    $\mu_{j^*} := \mu_{\text{prev}} - p_{\text{prev}} + h_{\text{prev}} + (1.0 - s_{j^*}) \cdot p_{j^*} - h_{j^*}$
    $c := c + w_{\text{prev}} - (1.0 - s_{j^*}) \cdot w_{j^*}$
    $s_{\text{prev}} := 0.0$; $s_{j^*} := 1.0$
    $\mathcal{M} := \mathcal{M} \setminus \{j^*\}$

    if $c > 0$ then
        while $j \notin \mathcal{M}$ do $j := j + 1$
        if $j \leq n$ then
        while $c - (1.0 - s_j) \cdot w_j \geq 0$ and $j \leq n$ do
            $\mu_{j^*} := \mu_{j^*} + (1.0 - s_j) \cdot p_j$
            $c := c - (1.0 - s_j) \cdot w_j$
            $s_j := 1.0$
            $j := j + 1$; while $j \notin \mathcal{M}$ and $j \leq n$ do $j := j + 1$
        if $j \leq n$ then
            $\mu_{j^*} := \mu_{j^*} + c/w_j \cdot p_j$
            $s_j := s_j + c/w_j$
            $c := 0.0$

    if $c < 0$ then
        while $j \notin \mathcal{M}$ do $j := j - 1$
        while $c + s_j \cdot w_j \leq 0$ (and $j \geq 1$) do
            $\mu_{j^*} := \mu_{j^*} - s_j \cdot p_j$
            $c := c + s_j \cdot w_j$
            $s_j := 0.0$
            $j := j - 1$; while $j \notin \mathcal{M}$ (and $j \geq 1$) do $j := j - 1$
        (if $j \geq 1$ then )
            $\mu_{j^*} := \mu_{j^*} + c/w_j \cdot p_j$
            $s_j := s_j + c/w_j$
            $c := 0.0$

    prev := $j^*$
```

Figure 2.1: Computing the μ_j upper bounds

Proposition 3. *No leading item between k and h can dominate leading item h.*

$$PKP_j \leq PKP_h \quad \forall k \leq j \leq h$$

Proof. First we observe that $\overline{x}$ is optimal for the binary knapsack problem with items in $[k,\dots,N]$ and it is also feasible for the binary knapsack problem with items in $[h,\dots,N]$; therefore it is optimal for all binary knapsack problems with items in $[j,\dots,N]$ for all j such that $k \leq j \leq h$. A second observation is that

$KP(h) = KP_h$: since $\overline{x}_h = 1$, the optimal solution of the binary knapsack problem in which all items in $[h, \dots, N]$ are available is also feasible for the binary knapsack problem in which item h is the selected leading item. Therefore it is also optimal for the latter problem. Now consider any index j such that $k \leq j \leq h$. For the two observations above and for relations (2.4), (2.5) and (2.7) we obtain $PKP_j = KP_j - p_j \leq KP(j) - p_j = KP(h) - p_j \leq KP(h) - p_h = KP_h - p_h = PKP_h$. $\Box$

Owing to Proposition 3 all leading items in the range $[k, \dots, h]$ can be discarded from further consideration without losing the optimality guarantee. A primal feasible solution to the PKP is given by $\overline{x}$ and its value is $KP(k) - p_h$.

In addition the information given by the binary knapsack problem is exploited to compute another upper bound ν_j^k to the optimal value of PKP_j for all $j > k$. This upper bound is defined as follows.

$$\nu_j^k = KP(k) - p_j \quad \forall j = k+1, \dots, N$$

Proposition 4. *The value ν_j^k is an upper bound to the value of the PKP with leading item j.*

$$\nu_j^k \geq PKP_j \quad \forall j = k+1, \dots, N$$

Proof. Consider any item $j > k$. For relations (2.4), (2.6) and (2.7) we obtain $\nu_j^k = KP(k) - p_j \geq KP(j) - p_j \geq KP_j - p_j = PKP_j$. $\Box$

In general there is no relation between the values of the two upper bounds μ_j and ν_j^k. The value of ν_j^k for any fixed k is monotonically non-decreasing when j increases, which is not true in general for the value of μ_j.

Proposition 5. *The upper bounds ν_j^k are non-decreasing with j for any fixed k.*

$$\nu_j^k \leq \nu_{j+1}^k \quad \forall j = k+1, \dots, N$$

Proof. The proof directly comes from inequality (2.5). $\Box$

After the computation of the optimal solution of a binary knapsack problem, the algorithm exploits Proposition 5 to sequentially test all candidate leading items $j > h$, starting from $h+1$ onward, and to discard them while ν_j^h remains below the value of the best incumbent solution, z_{LB}. This is equivalent to state that the algorithm discards all candidates $j > h$ whose penalty is not less than $KP(h) - z_{LB}$.

The pseudo-code of the algorithm is reported in Figure 2.2.

Optimization algorithm for the PKP

Input: For each item $j \in \mathcal{M}$ a weight a_j, a profit c_j and a penalty p_j; a capacity coefficient b.
Output: An optimal PKP solution x^* and a value z_{LB}

```
begin
  /* Initialization */
  $z_{LB}:= -\infty$; $l:= N$
  while $(\sum_{j=l}^{N} a_j \leq b)$ do
    if $(\sum_{j=l}^{N} c_j - p_l > z_{LB})$ then
      $z_{LB}:= \sum_{j=l}^{N} c_j - p_l$; $x_j^*:= 0\ \forall j < l$; $x_j^*:= 1\ \forall j \geq l$
    $l:= l - 1$
  $S:= \{1, \dots, l\}$
  /* Reduction */
  for each $j \in S$ do if $(c_j \leq p_j - p_{j+1})$ then $S:= S \setminus \{j\}$
  /* Compute upper bounds from linear relaxations */
  for each $j \in S$ do $\mu_j:= LKP_j - p_j$
  /* Examine all candidate leading items */
  repeat
    /* Select the most promising candidate */
    $k:= \text{argmax}_{j \in S}\{\mu_j\}$; $S:= S \setminus \{k\}$
    /* Termination test */
    if $(\mu_k \leq z_{LB})$ then goto end
    /* Solve a KP, store the optimal solution and its value */
    $\overline{x}:= \text{argmax}\{\sum_{i=k}^{N} c_i x_i : \sum_{i=k}^{N} a_i x_i \leq b, x_i \in \{0,1\}\ \forall i = k, \dots, N\}$
    $KP(k):= \max\ \{\sum_{i=k}^{N} c_i x_i : \sum_{i=k}^{N} a_i x_i \leq b, x_i \in \{0,1\}\ \forall i = k, \dots, N\}$
    for each $j = 1, \dots, k-1$ do $\overline{x}_j:= 0$
    /* Identify the leading item and apply the dominance criterion */
    $h:= k$
    while $(\overline{x}_h = 0)$ do
      $h:= h + 1$; $S:= S \setminus \{h\}$
    /* Update the best incumbent primal solution */
    if $(KP(k) - p_h > z_{LB})$ then
      $z_{LB}:= KP(k) - p_h$; $x^*:= \overline{x}$
    /* Compute the upper bound $\nu$ to discard more candidates */
    $j:= h + 1$
    while $(KP(k) - p_j \leq z_{LB})$ do
      $j:= j + 1$; $S:= S \setminus \{j\}$
  until $(S = \emptyset)$
end
```

Figure 2.2: Pseudo-code of our optimization algorithm for the PKP

2.4 Experimental analysis

Computing environment. Our algorithm was implemented in ANSI C. All the tests were done on a PC equipped with a Pentium IV 1.6 GHz processor, 512MB RAM and Linux operating system. In the absence of a competitor special-purpose code for the PKP, we tested our algorithm against the general-purpose solver CPLEX 8.1. A time limit of ten minutes was imposed for each problem instance.

Input data. To generate input data, we adapted the *gen2* generator, that was devised for the KP [72]. This generator receives in input the number of items

p type	Correlation	c type
p1	no correlation	c1
p2	weak correlation	c2
p3	strong correlation	c3
p4	inverse strong correlation	c4
p5	almost strong correlation	c5
p6	subset-sum problems	c6
p7	constant perimeter	
p8	constant area	
	profit = area	c7

Table 2.1: Summary of correlation types

(N), the range in which coefficients are generated (R), the method for generating weights (a type), the tightness of the capacity constraint (ρ), the type of correlation between penalties and weights (p type), the type of correlation between profits and weights (c type).

First we set $N = 1000$ and $R = 1000$ and we considered two different a types, indicated by *a1* and *a2*. In instances of class *a1* weights are generated at random with uniform probability distribution in $[1, \ldots, R]$; this corresponds to correlation type 1 in *gen2*. In instances of class *a2* weights are generated in the same way but small weights are discarded; this corresponds to correlation type 15 in *gen2*.

The parameter ρ is defined as the ratio between the capacity of the knapsack and the sum of the weights of all the items. We considered three different values for ρ, that is 0.5, 0.1 and 0.01.

Then we generated penalties in eight different ways (*p1...p8*) and profits in seven different ways (*c1...c7*) as shown in Table 2.1. In both cases the first six correlations correspond to those defined in *gen2* as correlation types numbered from 1 to 6. In *p7* instances we generated penalties setting $p_j = R - a_j + 1$, to represent constant perimeter rectangles, where the penalty is interpreted as the height of the rectangle. In *p8* instances we generated penalties setting $p_j = R/a_j$, to represent constant area rectangles. In *c7* instances we generated profits setting $c_j = p_j a_j$, to represent rectangular items whose profit is proportional to the area.

For each combination of the parameters 5 instances were generated. The overall number of instances in our dataset is therefore $2 \times 3 \times 8 \times 7 \times 5 = 1680$. Then we repeated our experiments, setting $N = 10000$ and $R = 10000$, thus generating 1680 more instances in the same way illustrated above.

Results and comments. The average results for each combination of parameters are reported in tables 2.2 and 2.3. Both tables represent the same results but grouped in different ways: in Table 2.2 the results are grouped according to the p correlation type; in Table 2.3 the results are grouped according to the c correlation type. Hence in Table 2.2 each row refers to seven different c correlation types (35 instances), while in Table 2.3 each row refers to eight different p correlation types (40 instances).

The first block of these tables indicates the a correlation type, the value of the parameter ρ and the p or c correlation type. The second block reports the results obtained by CPLEX: the average computing time and the number of instances solved to proven optimality within the time limit. The third block reports the results obtained by our algorithm: the average number of calls to the MINKNAP algorithm, the average computing time and the number of instances solved to proven optimality within the time limit. The number of calls and the computing time are referred only to the instances solved within the time limit. The forth block reports the results referred to the larger instances with 10000 items.

CPLEX could solve only 62.7% of the smaller instances to proven optimality within the time limit, while our algorithm solved all of them. In no case CPLEX was faster than our algorithm. For the classes in which both CPLEX and our algorithm solved the same instances, the latter was always faster. CPLEX was effective only for classes *c1* and *c2*, in which almost all instances were solved, but the running time was two orders of magnitude higher with respect to our algorithm. For these reasons only our algorithm was tested on larger instances.

The tightness of the capacity constraint and the type of correlation of weights affected the computational performances of both CPLEX and our algorithm. While for instances in class *a1* CPLEX performed better with tight capacity constraints (low values of ρ), the opposite was observed for instances of class *a2*. We explain this with the following observation: if the capacity constraints are tight, CPLEX is able to generate effective cover cuts but the upper bound given by the continuous relaxation of the knapsack problem may be loose; for random generation of the weights the former effect is stronger than the latter. When small weights are discarded, the latter effect is stronger than the former, since small weights help in filling the knapsack, and hence the integer optimal solution is more similar to the fractional optimal one. Our algorithm works better when the capacity constraint is tighter, even if the number of calls to the MINKNAP algorithm increases. In these cases a few items are selected in each integer KP solution and this makes the reduction test more effective.

An increase in the computation time for our algorithm occurred when solving instances of class *a2*, but the average number of calls to the MINKNAP algorithm was almost the same. Hence the loss of effectiveness was entirely due to a higher effort in solving the knapsack problem instances and this is consistent with observations made in other papers [72].

The hardest class for CPLEX was class *c4*: only 18.3% of the instances were solved. On the contrary class *c4* (inverse correlation) seems to be easier than class *c3* (direct correlation) for our algorithm.

From our experiments we conclude that the correlation between profits and penalties is more important than the correlation between penalties and weights.

When high profits may correspond to low penalties, the PKP solution is likely to be similar to the optimal solution of the KP obtained disregarding the penalties. On the opposite, when profits and penalties are correlated, the optimal solution of the knapsack problem tends to involve items with high profit, that are heavily penalized by high penalty coefficients. In this case the optimal solution of the KP and that of the PKP tend to be quite different.

Test on 2LSPP instances. Finally, we did some tests using our algorithm for pricing purposes in a branch-and-price algorithm for the 2LSPP. We considered two datasets proposed in the literature. The first one contains five classes of instances (named BENG, HGCUT, GCUT, HT, NGCUT); the second one contains ten classes of instances (named BW and MV). Both datasets are described in [68].

Root node. We considered what happens at the root node of a branch-and-price algorithm for the 2LSPP: we examined the task of solving the linear relaxation of the set covering reformulation of the 2LSPP with column generation and we compared two different pricing methods to be embedded in the column generation algorithm: the first is a general-purpose ILP solver (ILOG CPLEX) and the second is the algorithm presented in this paper. In table 2.4 we report the results of this experiment. The first block indicates the instance class name and the number of instances in each class. The second and third blocks include the number of iterations (CG iter.), the number of generated columns (cols) and the time required to complete the column generation algorithm. We set a time-out equal to one hour for each instance.

Our algorithm definitely outperformed CPLEX. On two of the ten instances in class BENG, the CPLEX-based column generation did not terminate in one hour (the corresponding values were discarded in the computation of the average results reported in column "time" of Table 2.4). On the remaining instances the CPU time required by CPLEX-based column generation is two orders of magnitude higher than that required by column generation using our algorithm.

2.5 Concluding remarks

In this paper we have introduced a penalized knapsack problem and we have presented an exact optimization algorithm. Although the problem has a compact ILP formulation, the computational results show that a general-purpose solver is not always effective. On the opposite our specialized algorithm proved to be much more effective and robust on a wide range of instances. Preliminary results show that the use of the algorithm presented in this paper as a pricing subroutine allows a branch-and-price algorithm to solve the 2LSPP faster than a state-of-the-art general-purpose ILP solver.

Instances			N = 1000					N = 10000		
			CPLEX 8.1		Our algorithm			Our algorithm		
a type	ρ	p type	time (s)	inst.	iter.	time (s)	inst.	iter.	time (s)	inst.
1	0.5	1	10.77	27	1.60	0.14	35	1.46	9.07	35
		2	42.54	24	2.51	0.09	35	6.19	39.30	32
		3	8.35	22	9.40	0.95	35	22.87	25.27	28
		4	9.42	24	12.06	1.84	35	25.60	40.92	29
		5	42.85	22	9.20	0.81	35	7.53	21.89	27
		6	14.96	22	13.86	1.54	35	38.85	70.00	27
		7	31.97	16	7.74	0.73	35	17.10	33.82	34
		8	13.71	26	2.57	0.38	35	1.97	18.84	32
1	0.1	1	5.74	28	2.00	0.08	35	3.42	30.94	34
		2	13.85	28	4.17	0.17	35	7.37	24.41	35
		3	15.03	20	13.06	0.90	35	15.21	14.56	29
		4	20.90	23	12.77	0.74	35	24.86	34.18	27
		5	6.45	18	8.69	0.47	35	37.81	92.23	31
		6	13.62	23	11.51	0.37	35	13.24	31.43	26
		7	9.12	25	12.77	1.46	35	9.57	8.95	32
		8	27.51	25	2.77	0.49	35	1.89	16.33	32
1	0.01	1	43.35	32	2.40	0.01	35	4.11	6.12	35
		2	4.79	30	3.17	0.05	35	6.84	17.48	34
		3	17.14	26	6.79	0.05	35	50.43	15.70	35
		4	7.72	25	11.40	0.11	35	32.29	10.11	31
		5	12.70	27	8.80	0.08	35	21.37	10.61	35
		6	10.35	29	8.49	0.07	35	28.31	7.02	35
		7	5.07	25	25.86	0.27	35	44.64	63.26	34
		8	8.54	22	3.20	0.09	35	2.26	7.04	35
2	0.5	1	17.42	23	2.34	0.54	35	2.06	45.07	35
		2	35.71	23	5.91	1.69	35	9.42	88.41	30
		3	7.50	20	17.51	3.14	35	7.63	52.59	28
		4	12.76	17	27.51	17.49	35	3.72	18.72	27
		5	8.05	19	14.60	5.54	35	16.34	57.23	31
		6	8.03	18	17.26	4.61	35	5.63	47.87	29
		7	13.98	24	12.06	2.30	35	1.40	12.99	30
		8	3.64	33	1.71	0.48	35	1.94	53.82	34
2	0.1	1	45.57	23	3.77	0.48	35	4.26	58.86	33
		2	56.61	20	8.37	1.05	35	9.22	84.07	30
		3	153.50	15	22.83	2.56	35	10.33	69.85	29
		4	56.52	15	18.06	2.14	35	5.73	31.04	25
		5	10.17	13	14.51	1.82	35	13.83	37.79	27
		6	14.79	16	21.09	1.91	35	19.92	50.98	27
		7	3.61	21	16.69	2.71	35	3.47	12.51	30
		8	4.43	32	2.17	0.42	35	1.88	42.17	33
2	0.01	1	24.88	35	4.14	0.02	35	8.80	30.64	35
		2	48.12	10	13.43	0.29	35	13.89	69.71	35
		3	8.39	10	61.26	2.57	35	50.10	78.66	29
		4	134.97	13	11.74	0.12	35	21.23	55.09	31
		5	18.99	11	47.06	1.80	35	43.44	71.44	30
		6	104.19	12	52.26	2.15	35	63.53	75.78	31
		7	181.84	11	37.34	0.35	35	5.92	44.35	25
		8	9.54	30	3.11	0.21	35	2.57	46.99	33

Table 2.2: Computational results for CPLEX 8.1 and our algorithm: penalties-weights correlation.

Problem			N = 1000					N = 10000		
			CPLEX 8.1		Our algorithm			Our algorithm		
a type	ρ	c type	time (s)	inst.	iter.	time (s)	inst.	iter.	time (s)	inst.
1	0.5	1	1.04	40	1.03	0.01	40	1.00	0.19	40
		2	5.15	40	1.08	0.01	40	1.05	0.19	40
		3	38.28	17	18.93	2.64	40	79.08	143.84	25
		4	10.15	8	7.53	1.00	40	15.46	53.52	36
		5	78.48	11	19.95	1.97	40	15.88	42.86	30
		6	26.39	38	1.58	0.01	40	1.33	0.30	40
		9	2.59	29	1.50	0.02	40	1.53	1.05	33
1	0.1	1	0.77	40	1.50	0.01	40	1.25	0.11	40
		2	4.35	40	1.33	0.01	40	1.21	0.12	39
		3	26.86	28	21.63	0.59	40	51.06	69.83	28
		4	24.62	12	15.33	2.18	40	10.59	75.73	33
		5	22.26	18	15.23	0.29	40	34.57	72.38	36
		6	13.52	37	2.28	0.02	40	2.95	3.40	40
		9	15.05	15	2.00	1.01	40	2.29	4.63	30
1	0.01	1	0.36	40	1.55	0.01	40	1.45	0.08	40
		2	0.88	40	1.39	0.01	40	1.40	0.08	40
		3	12.40	36	12.23	0.04	40	74.75	15.44	36
		4	19.16	10	26.73	0.52	40	32.99	63.81	38
		5	36.25	31	11.70	0.04	40	38.68	10.06	40
		6	16.41	38	5.63	0.02	40	13.88	25.96	40
		9	23.27	21	2.13	0.02	40	3.33	4.75	40
2	0.5	1	0.90	40	1.03	0.01	40	1.00	0.18	40
		2	15.98	40	1.03	0.01	40	1.00	0.19	40
		3	9.77	20	41.65	20.69	40	24.50	135.84	19
		4	9.17	5	11.95	2.76	40	3.14	124.98	34
		5	41.28	7	27.83	7.77	40	16.72	108.23	31
		6	24.35	33	2.03	0.02	40	1.98	1.22	40
		9	2.28	32	1.05	0.07	40	1.08	8.99	40
2	0.1	1	1.15	40	1.70	0.01	40	1.45	0.10	40
		2	15.54	40	1.45	0.01	40	1.23	0.12	40
		3	19.12	23	41.58	4.54	40	28.75	90.74	21
		4	104.42	3	17.20	3.47	40	7.14	177.34	26
		5	58.82	13	24.80	2.57	40	20.70	82.87	28
		6	144.24	18	5.80	0.09	40	4.93	5.25	40
		9	4.55	18	1.53	0.76	40	1.52	13.13	39
2	0.01	1	0.54	40	5.95	0.01	40	2.85	0.10	40
		2	62.83	37	5.25	0.02	40	2.33	0.11	40
		3	131.03	14	72.15	3.30	40	118.30	174.86	27
		4	14.52	6	20.03	0.60	40	10.46	117.40	33
		5	119.36	13	48.45	2.03	40	20.76	61.18	34
		6	51.86	13	44.20	0.49	40	29.74	59.82	35
		9	8.85	9	5.53	0.13	40	2.40	11.18	40

Table 2.3: Computational results for CPLEX 8.1 and our algorithm: profits-weights correlation

Problem		CPLEX 8.1			Our algorithm		
Class	# inst.	CG iter.	cols	time (s)	CG iter.	cols	time (s)
BENG	10	425.38*	435.38*	742.57*	524.60	927.40	7.42
CGCUT	3	70.00	82.00	5.94	57.33	108.33	0.06
GCUT	4	24.50	38.75	1.36	22.50	41.25	0.03
HT	9	86.67	90.89	1.89	63.44	117.44	0.04
NGCUT	12	24.50	29.83	0.25	17.50	36.33	0.01
Avg.		126.21	135.37	150.40	137.08	246.15	1.51
MV01	50	201.62	224.40	79.46	160.80	303.90	0.28
MV02	50	20.74	190.24	1.02	17.10	353.38	0.02
MV03	50	13.14	60.70	0.53	10.48	61.76	0.02
MV04	50	168.12	190.10	30.39	145.98	230.62	0.20
BW01	50	65.52	99.54	4.54	60.08	117.48	0.05
BW02	50	308.94	319.42	93.47	252.00	456.90	0.51
BW03	50	103.58	132.70	9.83	90.08	168.46	0.10
BW04	50	354.34	364.74	237.02	289.30	515.68	0.66
BW05	50	90.02	122.36	8.02	76.66	147.16	0.09
BW06	50	387.12	396.86	361.56	308.08	559.50	0.81
Avg.		171.31	210.11	82.58	141.06	291.48	0.27

Table 2.4: Comparison of pricing algorithms at the root node for the 2LSPP

Chapter 3

A branch-and-price algorithm for the two-dimensional level strip packing problem

The two-dimensional level strip packing problem (2LSPP) consists of packing rectangular items of given size into a strip of given width divided into horizontal levels. Items packed in the same level cannot be put on top of one another and their overall width cannot exceed the width of the strip. The objective is to accommodate all the items while minimizing the overall height of the strip. The problem is NP-hard and arises from applications in logistics and transportation. We present a set covering formulation of the 2LSPP amenable for a column generation approach, where each column corresponds to a feasible combination of items inserted into the same level. For the exact optimization of the 2LSPP we present a branch-and-price algorithm, where the pricing problem is a penalized knapsack problem. Computational results are reported for benchmark instances with some hundreds items.

3.1 Introduction

In several industrial applications it is required to place a set of rectangular items in standard stock units. In wood and glass manufacturing, for instance, rectangular components must be cut from large pieces of material; in warehouses, the goods must be placed on shelves; in the design of newspapers' layout it is needed to arrange in an effective way articles and advertisements. This kind of applications are often modeled as two-dimensional packing or cutting problems. Packing and cutting is a wealthy research area: a complete review on related models and methods can be found in [30], while in [35], [36] and [34] the authors propose general graph-theoretical frameworks for devising bounds on multi-dimensional packing problems.

In production contexts as clothes or paper manufacturing, a single strip of material is often available, and a set of items must be obtained from the strip. The aim is to pack the items minimizing the height of the portion of strip to be cut. This problem is called two-dimensional strip packing (2SPP). Recently a fully polynomial time approximation scheme for the 2SPP has been proposed [56]. Metaheuristic approaches include simulated annealing [32] and genetic algorithms [50]. In [71] a branch-and-bound algorithm is presented, for the exact optimization of the 2SPP, that is able to solve instances with up to 200 items in one hour of computing time.

We study a variation of the 2SPP in which a further restriction is imposed: the items must be organized into horizontal strips, indicated as *levels*; inside each level, the items cannot be put on top of one another. This kind of variation, referred to as two-dimensional level strip packing (2LSPP) [67], is $\mathcal{NP}$-hard in the strong sense, since it contains the bin-packing problem as a special case [41].

The 2LSPP can be approximated with fast heuristics, which provide also an a priori guarantee on the quality of the solution [12] [67]. More recently, Lodi et al. [67] proposed a formulation for the 2LSPP involving a polynomial number of variables and constraints; the effectiveness of state-of-the-art general purpose ILP solvers makes this approach particularly appealing.

In this paper we introduce a new formulation for the 2LSPP as a set covering problem. The linear relaxation of this model is computed with column generation, and the bound found in this way is used in a branch-and-price enumeration algorithm. In Section 3.2 we present our formulation, and we discuss its relationships with the compact formulation of Lodi et al.; we also describe our method for solving the corresponding linear program. In Section 3.3 we highlight the main issues in the design of our branch-and-price algorithm. Finally, in Section 3.4 we report the details of an experimental analysis.

3.2 Problem formulation

In the 2LSPP it is given a strip, whose width is a positive integer W, and a set $\mathcal{N}$, where each $j \in \mathcal{N}$ is an item whose width and height are a positive integers denoted with w_j and h_j respectively. The items must be organized into horizontal strips called levels: the sum of widths of items in the same level cannot exceed the width of the strip.

Items in the same level cannot be piled up. Therefore, the height of each level corresponds to the maximum height of an item in that level. We call this particular item the *leading item*, and we say that the leading item *initializes* the level. Through the paper we assume that the items are sorted by non-decreasing height values: $h_i \leq h_j$ for each $i < j$. Hence, without loss of generality, we can state that no item $i > j$ can be assigned to a level initialized by j. Lodi et al. [67] proposed the following compact formulation for this problem:

$$\min \sum_{j \in \mathcal{N}} h_j x_{jj} \tag{3.1}$$

$$\text{s.t.} \sum_{j \geq i} x_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{3.2}$$

$$\sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj} \qquad \forall j \in \mathcal{N} \tag{3.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \leq j \in \mathcal{N} \tag{3.4}$$

Each binary variable x_{ij} indicates whether item i is assigned to a level in which j is the leading item; therefore each binary variable x_{jj} indicates whether item j is a leading item. Because of the ordering of the items, we can fix each x_{ij} variable with $i > j$ to 0 and remove it from the model. Constraints (3.2) impose that each item is assigned to a level. Constraints (3.3) impose that the sum of widths of the items assigned to the same level does not exceed the width of the strip. The objective is to minimize the overall height of the strip.

3.2.1 A set covering reformulation

A lower bound for the 2LSPP can be obtained from the model above by neglecting the integrality conditions (3.4). We sharpen this bound exploiting Dantzig-Wolfe decomposition [76]: let Ω_j be the set of levels respecting the width constraints, defined as follows for each $j \in \mathcal{N}$:

$$\Omega_j = \{x | \sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj}, 0 \leq x_{ij} \leq 1\}.$$

Let K_j be the set of the integer points in Ω_j and let x^k be the generic integer point of Ω_j. Each point x in the convex hull of Ω_j can be expressed as a convex combination of the integer points in K_j:

$$\text{conv}(\Omega_j) = \{x | x = \sum_{k \in K_j} x^k z_k, \sum_{k \in K_j} z_k = 1 \quad \text{and} \quad 0 \leq z_k \leq 1. \tag{3.5}$$

Hence, by substitution from the linear relaxation of the 2LSPP, a relaxation of the 2LSPP is the following:

$$
\begin{aligned}
\min \; & \sum_{j \in \mathcal{N}} h_j \sum_{k \in K_j} x_j^k z_k & & \\
\text{s.t.} & \sum_{j \geq i} \sum_{k \in K_j} x_i^k z_k = 1 & & \forall i \in \mathcal{N} \\
& \sum_{k \in K_j} z_k = 1 & & \forall j \in \mathcal{N} \qquad (3.6) \\
& 0 \leq z_k \leq 1 & & \forall j \in \mathcal{N}, \forall k \in K_j
\end{aligned}
$$

Here, all polyhedra Ω_j have been replaced by their convex hulls. Due to this convexification, the bound found by optimizing this model is at least as tight as that of the linear relaxation of the 2LSPP.

Each K^j contains a point representing an empty level: they can be dropped and considered implicitly by stating constraints (3.6) as inequalities:

$$
\begin{aligned}
\min \; & \sum_{j \in \mathcal{N}} h_j \sum_{k \in K_j} z_k & & \\
\text{s.t.} & \sum_{j \geq i} \sum_{k \in K_j} x_i^k z_k = 1 & & \forall i \in \mathcal{N} \qquad (3.7) \\
& \sum_{k \in K_j} z_k \leq 1 & & \forall j \in \mathcal{N} \qquad (3.8) \\
& 0 \leq z_k \leq 1 & & \forall j \in \mathcal{N}, \forall k \in K_j
\end{aligned}
$$

Since in all optimal solutions no item is chosen more than once as a leading item, constraints (3.8) are redundant and can be deleted. Furthermore, the set partitioning constraints (3.7) can be replaced by set covering constraints, because it is never convenient to pack an item in more than one level.

The resulting model is the following:

$$
\begin{aligned}
\text{MP)} \min \; & \sum_{j \in \mathcal{N}} h_j \sum_{k \in K_j} z_k & & \qquad (3.9) \\
\text{s.t.} & \sum_{j \geq i} \sum_{k \in K_j} x_i^k z_k \geq 1 & & \forall i \in \mathcal{N} \qquad (3.10) \\
& 0 \leq z_k \leq 1 & & \forall j \in \mathcal{N}, \; \forall k \in K_j \qquad (3.11)
\end{aligned}
$$

In this master problem (MP) the column corresponding to each variable z_k with $k \in K_j$ represents a feasible set of items packed into a same level j. An alternative formulation of the 2LSPP is obtained by restoring the integrality conditions $z_k \in \{0, 1\}$.

3.2.2 The pricing problem

Model (3.9) – (3.11) may have a huge number of columns. Therefore, a problem involving a restricted set of variables (RMP) is considered and columns not included in the RMP are iteratively generated when needed, in order to obtain an optimal solution of the whole MP.

Let λ be the vector non-negative dual variables associated to covering constraints (3.10) in a RMP optimal solution. The pricing problem we need to solve to identify new columns is the following: $\pi(\lambda) = \min \ _j\{\pi_j(\lambda)\}$, where for each $j \in \mathcal{N}$

$$\begin{aligned}
\pi_j(\lambda) = \min \quad & h_j x_j - \sum_{j \in \mathcal{N}} \sum_{i \le j} \lambda_i x_i \\
\text{s.t.} & \sum_{i<j} w_i x_i \le (W - w_j) x_j && \forall j \in \mathcal{N} \\
& x_i \in \{0,1\} && \forall j \in \mathcal{N},\ \forall i \le j
\end{aligned}$$

Thus a negative reduced cost column can be generated by solving at most $|\mathcal{N}|$ binary knapsack problems, obtained by setting to 1 one x_j variable at a time.

However solving a large number of knapsack problems to optimality to generate negative reduced cost columns can be unnecessary, since we just need one negative reduced cost column, provided it exists. Therefore we solve a pricing problem in which the leading item is not fixed, but rather it must be chosen in an optimal way, that is we search for the column of minimum reduced cost for all possible choices of the leading item. The pricing problem can be rewritten in an equivalent way as follows:

$$\begin{aligned}
\pi(\lambda) = \min \quad & \eta - \sum_{i \in \mathcal{N}} \lambda_i x_i && && (3.12) \\
\text{s.t.} & \sum_{i \in \mathcal{N}} w_i x_i \le W && && (3.13) \\
& h_i x_i \le \eta && \forall i \in \mathcal{N} && (3.14) \\
& x_i \in \{0,1\} && \forall i \in \mathcal{N}
\end{aligned}$$

Each binary variable x_i is equal to 1 if and only if item i is assigned to the level. The free variable η is a penalty term. Only one of the constraints (3.14) is active at a time. That is, the value of η is determined by the height of the leading item of the level. The capacity constraint (3.13) still impose that the overall width of the level does not exceed the width of the strip.

The objective function (3.12) can be stated in maximization form

$$\pi(\lambda) = -\max \quad \{\sum_{i \in \mathcal{N}} \lambda_i x_i - \eta\}.$$

This pricing problem can be solved with special purpose algorithms for the penalized knapsack problem (PKP), discussed in Chapter 2.

3.3 Branch-and-price

Branching strategy.

We base our branching rule on the x variables of the compact formulation instead of considering the z variables of the MP: once an optimal MP solution z^* is obtained, a corresponding (fractional) solution x^* in terms of the original variables can be found by fixing $x^*_{ij} = \sum_{j \geq i} \sum_{k \in K_j} x^k_i z^*_k$ for each $i, j \in \mathcal{N}$.

We have adopted a two-stage branching strategy: in the first stage search tree branching decisions are taken on the x_{jj} variables, that is the leading items are chosen; since in each leaf of this search tree the set of leading items, and therefore the height of each level, is defined, the aim of the second stage search tree is to solve a feasibility problem: the remaining items must be organized in the levels initialized by the chosen leading items, without violating the width and height constraints. In both stages, branching is operated on the variable whose x^*_{ij} value is closest to 0.5. This variable is fixed to 0 in one branch and to 1 in the other branch.

These conditions slightly change the structure of the pricing problem. On the first stage, each time a x_{jj} variable is fixed to 1, j is discarded from the set of items in the PKP optimization, and an additional KP is solved, for computing the best completion of the solution in which j is the leading item; when a x_{jj} variable is fixed to 0, it is simply discarded from the set of candidate leading items in the PKP computation. Since all the x_{jj} variables are fixed on the first stage search tree, on the second stage one has to solve a KP for each of the chosen leading items. Therefore, the fixing of further x_{ij} variables only reduces the dimension of these KP instances.

The search trees are explored in a best-bound-first order.

Initialization. In order to obtain an initial set of columns to populate the RMP, we use the well known Best-Fit Decreasing-Height (BFDH) heuristic [67]. The items are iteratively considered from item $|\mathcal{N}|$ down to item 1 and in each iteration the current item is packed into the level with the minimum residual capacity among those which can receive it. If the item cannot be accommodated in this way, a new level is initialized. We implemented a simple randomized version of this heuristic (r-BFDH): a preprocessing step is added, in which a fixed number r of items are randomly drawn from a uniform probability distribution and the corresponding levels are initialized.

Besides running the original version of BFDH once, three r-BFDH solutions are computed for each r value from 1 to $\lceil \sum_{i \in \mathcal{N}} w_i/W \rceil$, that is the number of levels in a fractional solution rounded up to the nearest integer; this is a lower bound on the number of levels composing an optimal solution. The best BFDH solution value found in this way is also kept as an initial upper bound.

Upper bounds. We experimentally observed that the r-BFDH heuristic often provides tight bounds. Nevertheless, we incorporate a fast rounding heuristic for the set covering problem, in order to search for good integer solutions during the exploration of the search tree. This works as follows: initially, all the items are uncovered, and the columns of the RMP are sorted by non-increasing value of the corresponding z_k variables; following this order, each column k is considered: if column k represents a level containing uncovered items, the corresponding z_k variable is rounded up to 1 and each item in k is marked as covered, otherwise the z_k variable is fixed to 0.

We run this heuristic once for each node of the search tree, when the column generation process is over.

Problem reduction. Consider a generic node $\mathcal{P}$ of the search tree; let $v(\mathcal{P})$ be the value of the partial solution identified by $\mathcal{P}$, $\mathcal{N}(\mathcal{P})$ be the set of items fixed as leading items in that partial solution, and UB be the value of the best incumbent integer solution. For each item $j \in \mathcal{N} \setminus \mathcal{N}(\mathcal{P})$, if $v(\mathcal{P}) + h_j \geq UB$, then j can be discarded from the set of candidate leading items in node $\mathcal{P}$, since by fixing j to 1 the node would be fathomed.

Columns deletion and re-insertion. We found useful to periodically remove unpromising columns from the RMP: each time a node of the search tree is considered, the columns in the RMP whose reduced costs are higher than a threshold are shifted into a separate pool. The reduced cost of each column is computed with respect to the optimal dual solution on the ancestor node. In our implementation, the removal threshold is computed as the difference between the best known upper and lower bounds, divided by $\lceil \sum_{i \in \mathcal{N}} w_i/W \rceil$.

The columns pool is scanned at each column generation iteration: whenever a column is found, whose reduced cost is negative with respect to the current dual solution, it is re-inserted in the RMP. Each column is kept into the pool for up to 6 checks.

Lagrangean bounds. It is well known that the bound obtained by optimizing the set covering linear program can also be obtained by solving a Lagrangean dual problem [76], when the set of constraints (3.2) are relaxed:

$$
\begin{aligned}
\max_{\lambda} \ \omega(\lambda) = \min \ & \sum_{j\in\mathcal{N}} h_j x_{jj} - \sum_{i\in\mathcal{N}} \lambda_i \Big(\sum_{j\geq i} x_{ij} - 1\Big) & \\
\text{s.t.} & \sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj} & \forall j \in \mathcal{N} \\
& x_{ij} \in \{0,1\}. & \forall i \leq j \in \mathcal{N}
\end{aligned}
$$

For each set of multipliers λ, this problem is analogous to the pricing problem for the set covering formulation of the 2LSPP. In fact, it decomposes into independent subproblems, one for each $j \in \mathcal{N}$:

$$
\begin{aligned}
\min \quad & h_j x_{jj} - \sum_{i\leq j} \lambda_i x_{ij} & \\
\text{s.t.} & \sum_{i<j} w_i x_{ij} \leq (W - w_j) x_{jj} & \forall j \in \mathcal{N} \\
& x_{ij} \in \{0,1\}. & \forall i \leq j \in \mathcal{N}
\end{aligned}
$$

Therefore, each subproblem j can be optimized by considering two cases: if variable x_{jj} is fixed to 1, then the remaining problem is a binary knapsack; this is solved to optimality obtaining a value $\pi_j(\lambda)$. If variable x_{jj} is fixed to 0, then each variable x_{ij} with $i < j$ must be set to 0; this yields a solution of value 0. Hence, for any choice of the λ_i multipliers, a valid lower bound $\omega(\lambda)$ for 2LSPP is given by:

$$
\omega(\lambda) = \sum_{i\in\mathcal{N}} \lambda_i + \sum_{j\in\mathcal{N}} \min \ \{\pi_j(\lambda), 0\}
$$

However, a key property of our pricing routine is actually to implicitly consider these π_j values to avoid the computation of a large number of knapsack problems. In fact, the one with minimum value is computed by solving a PKP. Therefore, a lower bound $\bar{\omega}(\lambda)$ on $\omega(\lambda)$ can be obtained by substituting any $\pi_j(\lambda)$ value with a corresponding lower bound $\bar{\pi}_j(\lambda)$.

$$
\bar{\omega}(\lambda) = \sum_{i\in\mathcal{N}} \lambda_i + \sum_{j\in\mathcal{N}} \min \ \{\bar{\pi}_j(\lambda), 0\}
$$

We initially approximate each $\bar{\pi}_j(\lambda)$ with the value of the linear relaxation of the corresponding subproblem. These values are readily available, since they are computed in a preprocessing step by the algorithm for the PKP. Furthermore, whenever a tighter bound is computed during the optimization of the PKP, the corresponding $\bar{\pi}_j(\lambda)$ value is updated and the quality of the $\bar{\omega}(\lambda)$ bound improved.

Whenever, during the column generation iterations, the difference between the highest $\bar{\omega}(\lambda)$ value encountered and the RMP optimal value is less than 10^{-6}, the column generation process is terminated, and such Lagrangean bound is kept as lower bound.

Variable Fixing with Lagrangean Penalties: We also used the $\bar{\pi}_j(\lambda)$ values in a variable fixing procedure. Once these values have been computed, the following reduction tests can be checked in linear time: let UB be the value of the incumbent integer solution,

- for each j such that $\bar{\pi}_j(\lambda) < 0$, if $\lceil \bar{\omega}(\lambda) - \bar{\pi}_j \rceil \geq UB$ then j can be fixed as a leading item ($x_{jj} = 1$),
- for each j such that $\bar{\pi}_j(\lambda) > 0$, if $\lceil \bar{\omega}(\lambda) + \bar{\pi}_j \rceil \geq UB$ then j can be discarded from the set of candidate leading items ($x_{jj} = 0$),

since in both cases, the opposite choice would push the lower bound above the upper bound, causing the fathoming of the node.

Combinatorial bound. Finally, we incorporated in our bounding procedure a combinatorial bound (CUT in the remainder) proposed by Lodi et al. [68]. It consists in splitting each item in vertical strips of width 1, and building levels by considering these strips in order of non-increasing height. This bound dominates that given by the LP relaxation of the compact formulation, but no relation of dominance exists with the set covering LP bound. Since we are assuming that such a sorting of the items is carried out in a preprocessing step, this bound can be computed in linear time.

The CUT bound is computed, for each node of the search tree, before the column generation process is started. Whenever the value of an RMP optimal solution is found to be less than the value of the CUT bound, the column generation process is halted, and the CUT bound is kept as a lower bound.

3.4 Computational results

Our branch-and-price algorithm was implemented in C++, and compiled with a GNU C/C++ compiler version 3.2.2. We solved the restricted linear programs with the CPLEX 8.1 implementation of the primal simplex algorithm. All the internal CPLEX parameters are kept at their default values. All our experiments were run on a Linux workstation equipped with a Pentium IV 1.6GHz processor and 512MB of RAM. A time limit of 1 hour was imposed to each test. Furthermore, the program was halted whenever the computation exceeded the amount of physical memory.

In order to assess the effectiveness of our method, we considered two datasets for two-dimensional packing problems widely used in the literature; they are both described in [68]. The first one consists of 5 classes of instances: BENG (10 instances), CGCUT (3 instances), GCUT (4 instances), HT (9 instances) and NGCUT (12 instances). The second dataset consists of 500 instances, divided into 10 classes of 50 instances, named MV and BW. They contain instances involving up to 200 items, with different types of correlation between height and width of the items.

Lower bounds. First, we measured the quality of the set covering LP bound (SC bound), by comparing it with the LP relaxation of the compact formulation (LP bound) and the CUT bound. As a measure of duality gap we took, for each instance $(UB - LB)/UB$, where UB is the value of the BFDH heuristic solution and LB is in turn the value of SC, LP and CUT bounds. In tables 3.1(a) and 3.1(b) we report the average values for the instances in each class of the first and second dataset respectively. Each row of these tables corresponds to a class of instances, except for the last one that contains the overall average values. Each column refers to a bounding strategy, except for the first one that contains the class identifiers.

The LP bound is weaker than the CUT bound also from an experimental point of view. It is therefore not competitive. On the instances of the first dataset, CUT is on the average the tightest bound, while on the instances of the second dataset the SC bound is clearly superior (see, for instance, classes BW03 and BW05). On the other hand, the computation of the SC bound is two orders of magnitude slower than that of the CUT bound.

Finally, it is worth noting that the CUT and SC bounds seem to be complementary, since they yield good approximations in different classes of instances. This observation was one of the motivations for including the computation of both bounds in a unique routine, when tackling the problem of solving 2LSPP to optimality.

Solving the 2LSPP to proven optimality. We compared the performance of our branch-and-price with that of CPLEX 8.1, used as a general purpose ILP solver to optimize the compact model (3.1)–(3.4). Tables 3.2(a) and 3.2(b) contain the results for the first and second dataset respectively. As before, each entry of the table represents an average value between the instances in a class, and the class identifiers are indicated in the first column. Each table is composed by two horizontal blocks; each of them corresponds to the solution method indicated

Class	LP bound	CUT bound	SC bound
BENG	6.75%	0.47%	4.69%
GCUT	14.91%	10.57%	0.14%
NGCUT	12.57%	4.21%	5.10%
CGCUT	4.66%	4.66%	6.95%
HT	7.80%	0.40%	4.09%
Avg.	9.34%	4.06%	4.19%

(a)

Class	LP bound	CUT bound	SC bound
MV 01	8.73%	6.37%	2.29%
MV 02	7.80%	1.00%	5.46%
MV 03	11.95%	8.97%	2.96%
MV 04	7.99%	1.55%	3.80%
BW 01	11.90%	9.40%	2.19%
BW 02	8.68%	1.79%	3.78%
BW 03	14.57%	12.17%	0.69%
BW 04	8.61%	5.42%	4.53%
BW 05	19.18%	17.77%	0.04%
BW 06	9.24%	4.80%	2.56%
Avg.	10.87%	6.92%	2.83%

(b)

Table 3.1: Comparison of lower bounds

in the first row. In each block we report the number of instances in each class solved to proven optimality (solved inst.), the average gap between the value of the incumbent primal solution (UB) and the global lower bound (LB) on the instances whose optimality was not proven, computed as (UB - LB) / UB, and the average computing time on the instances that were closed. In the last row of each table we report the total number of instances solved to proven optimality.

Branch-and-price solves all the instances in the first dataset, while CPLEX leaves a large gap on 7 of the 10 BENG instances. Moreover, branch-and-price is on the average much faster on the remaining classes. Branch-and-price performs much better than CPLEX also on the instances of the second dataset, solving more problems and consistently giving better computing time or tighter approximations.

As a final assessment, we tried to turn off the CUT bound computation in our procedure, in order to test the effectiveness of a pure column generation routine. We observed that, besides its contribution in making the computation faster, the CUT bound is determinant only on the BENG instances of the first dataset. This is further confirmed by comparing the results regarding the quality of bounds with those regarding the optimization process: instances MV01, MV03, BW01, BW03 and BW05 can be treated well by branch-and-price, althought the CUT bound is experimentally loose.

Class	B&P solved inst.	avg. gap	time(s)	CPLEX 8.1 solved inst.	avg. gap	time(s)
BENG	10	0	0.02	3	4.96%	24.47
GCUT	4	0	3.18	4	0	1.06
NGCUT	12	0	0.02	12	0	0.09
CGCUT	3	0	0.43	3	0	67.01
HT	9	0	0.02	9	0	10.14
	38			31		

(a)

Class	B&P solved inst.	avg. gap	time(s)	CPLEX 8.1 solved inst.	avg. gap	time(s)
MV 01	48	0.68%	12.35	48	0.54%	14.09
MV 02	48	0.99%	6.51	25	4.26%	150.01
MV 03	49	0.26%	8.30	47	0.47%	22.49
MV 04	41	1.03%	55.03	21	3.98%	173.26
BW 01	49	0.28%	2.91	48	0.60%	19.29
BW 02	36	1.00%	202.71	21	4.35%	114.12
BW 03	50	0.00%	0.23	50	0.00%	0.16
BW 04	23	1.31%	123.81	17	2.08%	94.71
BW 05	50	0.00%	0.08	50	0.00%	0.04
BW 06	43	1.20%	99.33	34	1.20%	224.74
	437			361		

(b)

Table 3.2: Solving the 2LSPP to proven optimality

Chapter 4

An optimization algorithm for the ordered open-end bin packing problem

The ordered open-end bin packing problem is a variant of the bin packing problem in which the items to be packed are sorted in a given order and the capacity of each bin can be exceeded by the last item packed into the bin. In this work we present a branch-and-price algorithm for its exact optimization. The pricing problem is a special variant of the binary knapsack problem, in which the items are ordered and the last one does not consume capacity. We present a specialized optimization algorithm for this subproblem. We also discuss the effectiveness of different branching rules and other implementation details of our branch-and-price algorithm. Computational results are presented on instances of different size and items with different correlations between their size and their position in the given order.

4.1 Introduction

The open-end bin packing problem is a is a variant of the bin packing problem [23] in which the capacity of each bin can be exceeded by the last item packed into the bin. This problem was introduced in a paper by Yang and Leung [66]. We study a variation of this problem introduced in [103], called ordered open-end bin packing problem (OOEBPP) in which the packing must respect a given order of the items. The motivation given by the authors for studying this problem is related to the fare payment in subway stations in Hong Kong. Yang and Leung examined several algorithms for on-line and off-line approximation and studied their worst-case and average-case performance.

In this work we present a branch-and-price algorithm for the exact optimization of the OOEBPP. In Section 4.2 we introduce the notation used in the paper and we give a compact formulation of the problem; then we present a set covering reformulation of the OOEBPP, we introduce a combinatorial bound and we show how to derive a set of valid inequalities, that may strengthen the set covering reformulation. Since the set covering formulation involves an exponential number of variables, in Section 4.3 we present a specialized procedure for generating columns dynamically. The so-called pricing problem, in fact, is a special variant of the binary knapsack problem, in which the items are ordered and the last one does not consume capacity. We present a specialized optimization algorithm for this subproblem, that allows to effectively solve the pricing problem to optimality, exploiting suitable bounds and domination criteria. In Section 4.5 we conclude our description of the branch-and-price algorithm by describing primal bounding procedures and some implementation details. We discuss also how to exploit dual solutions to obtain faster convergence and designing variable fixing procedures. In Section 4.6 computational results are presented on instances of different size and items with different correlation between their size and their position in the given order.

4.2 Problem formulation

The OOEBPP is defined as follows. An ordered sequence $\mathcal{N}$ of items is given; each item $j \in \mathcal{N}$ has a given positive integer weight a_j. The items must be packed into identical bins with a given positive integer capacity b. The objective is to minimize the number of bins, with the constraint that the capacity of each bin can be exceeded only by the last item packed into it, where the term "last" is referred to the ordering of the items in $\mathcal{N}$. In the remainder we call such item the *overflow item* of its bin, and we say that it *initializes* its bin. Through the paper, we suppose that each element of $\mathcal{N}$ is identified by a positive integer, that is $\mathcal{N} = \{1 \dots N\}$. Therefore, we assume that each item $j \in \mathcal{N}$ and the j-th item of $\mathcal{N}$ coincide. An ILP formulation of the problem is the following.

$$\text{minimize} \sum_{i \in \mathcal{N}} y_i \tag{4.1}$$

$$\text{s.t.}\, y_i + \sum_{j>i} x_{ij} = 1 \qquad \forall i \in \mathcal{N} \tag{4.2}$$

$$\sum_{i<j} a_i x_{ij} \leq (b-1) y_j \qquad \forall j \in \mathcal{N} \tag{4.3}$$

$$x_{ij} \in \{0,1\} \qquad \forall i < j \in \mathcal{N} \tag{4.4}$$

$$y_i \in \{0,1\} \qquad \forall i \in \mathcal{N} \tag{4.5}$$

Each binary variable y_i indicates whether item i is the overflow item in its bin. Hence the number of bins used is indicated in the objective function (4.1) by the number of binary variables y_i set to 1. Each binary variable x_{ij} indicates whether item i is assigned to the bin in which the overflow item is item j. Because of the constraint on the ordering of the items, we have x_{ij} variables with $i < j$ only. Constraints (4.2) impose that each item is assigned to a bin, while capacity constraints (4.3) impose that the overall weight of the items assigned to a bin, excluding the overflow item, must fit into the bin and must leave at least one capacity unit available for accommodating the overflow item.

The problem is $\mathcal{NP}$-hard [66].

4.2.1 A set covering reformulation

A lower bound for OOEBPP can be obtained from the linear relaxation of (4.1)–(4.5), where integrality conditions (4.4) and (4.5) are neglected. Instead, the branch-and-price algorithm we present in this paper relies on a set covering reformulation of the OOEBPP. Consider the set Ω_j defined as follows for each $j \in \mathcal{N}$:

$$\Omega_j = \{(x_{ij}, y_j) | \sum_{i<j} a_i x_{ij} \leq (b-1) y_j, 0 \leq x_{ij} \leq 1, 0 \leq y_j \leq 1\}.$$

Let K_j be the set of the integer points in Ω_j and let $(x_{ij}, y_j)^k$ be the generic integer point of Ω_j. Then each point (x_{ij}, y_j) in the convex hull of Ω_j can be expressed as a convex combination of the integer points in K_j:

$$(x_{ij}, y_j) = \sum_{k \in K_j} (x_{ij}, y_j)^k z_k \tag{4.6}$$

with $\sum_{k \in K_j} z_k = 1$ and $0 \leq z_k \leq 1$. Exploiting equation (4.6) we obtain by substitution the following reformulation of the linear relaxation of the OOEBPP, where all polyhedra Ω_j have been replaced by their convex hulls:

$$\begin{aligned}
\text{minimize} & \sum_{j\in\mathcal{N}}\sum_{k\in K_j} y_j^k z_k & & \\
\text{s.t.} & \sum_{k\in K_i} y_i^k z_k + \sum_{j>i}\sum_{k\in K_j} x_{ij}^k z_k = 1 & & \forall i\in\mathcal{N} \\
& \sum_{k\in K_j} z_k = 1 & & \forall j\in\mathcal{N} \qquad (4.7)\\
& 0\le z_k \le 1 & & \forall j\in\mathcal{N}, \forall k\in K_j
\end{aligned}$$

Excluding from this linear program all columns corresponding to the integer points in which all variables are zero (those with $y_j^k = 0$, that implies $x_{ij}^k = 0$ for each $i < j$), constraints (4.7) can be rewritten as inequalities:

$$\begin{aligned}
\text{minimize} & \sum_{j\in\mathcal{N}}\sum_{k\in K_j} z_k & & \\
\text{s.t.} & \sum_{k\in K_i} z_k + \sum_{j>i}\sum_{k\in K_j} x_{ij}^k z_k = 1 & & \forall i\in\mathcal{N} \qquad (4.8)\\
& \sum_{k\in K_j} z_k \le 1 & & \forall j\in\mathcal{N} \qquad (4.9)\\
& 0\le z_k \le 1 & & \forall j\in\mathcal{N}, \forall k\in K_j
\end{aligned}$$

Now we observe that in all optimal solutions no item will be chosen more than once as the overflow item of a bin. Therefore constraints (4.9) are redundant and can be deleted. Hence the remaining model only contains set partitioning constraints (4.8): in turn these can be replaced by set covering constraints, because it is never convenient to pack an item more than once. So we obtain the following set covering model, which is at least as tight as the linear relaxation of the OOEBPP, owing to the convexification of constraints (4.3):

$$\begin{aligned}
\text{minimize} & \sum_{j\in\mathcal{N}}\sum_{k\in K_j} z_k & & \qquad (4.10)\\
\text{s.t.} & \sum_{k\in K_i} z_k + \sum_{j>i}\sum_{k\in K_j} x_{ij}^k z_k \ge 1 & & \forall i\in\mathcal{N} \qquad (4.11)\\
& 0\le z_k \le 1 & & \forall j\in\mathcal{N},\ \forall k\in K_j \qquad (4.12)
\end{aligned}$$

In this reformulated model each variable $z_k, k \in K_j$ corresponds to a feasible column, that is a feasible set of items packed into a same bin j.

4.2.2 A combinatorial lower bound

The linear relaxation of the model (4.10) - (4.12) can be strengthened by valid inequalities, which also provides a valid lower bound.

Consider the last element of the ordered sequence $\mathcal{N}$; it is must obviously be the overflow item of its bin. Consider now item $N-1$; there are two cases: either it fits into the same bin initialized by item N or it must initialize another bin. The same argument can be repeated for each item down to the beginning of $\mathcal{N}$. Whenever the residual capacity left by items $j+1, \ldots, N$ is not enough to accommodate item j, a new bin must be initialized. The overflow item of the new bin is selected as the one with maximum size among those in $j, \ldots, N$ and not yet chosen as overflow items. After N iterations this procedure, called CB algorithm (for Combinatorial Bounding) in the remainder, returns a valid lower bound to the number of necessary bins. The pseudo-code of the procedure is reported in Figure 4.1. If the set denoted with S in the pseudo-code is implemented with a heap data structure, the complexity of the procedure is $O(NlogN)$. It is not hard to note that this combinatorial bound, although being simple and fast, dominates the bound given by the LP relaxation of (4.1)–(4.5). In fact, the solution given by CBA is feasible for this LP, but can obviously be non-optimal.

Since the CB algorithm considers the overall residual capacity rather than the residual capacity of each bin, the lower bound may have no correspondence with any feasible integer solution, because the items cannot be splitted. However it is useful in two different ways: first, it provides a lower bound to be exploited in the branch-and-price algorithm to prune the search tree; second, it yields a set of items in correspondence of which the need of using an additional bin has been detected. These are put in the set $\mathcal{B}$ in the pseudo-code.

This piece of information allows to strengthen the linear relaxation of the master problem, by the following valid inequalities:

$$\sum_{k \in \cup_{i \geq \mathcal{B}(t)} K_i} z_k \geq t \tag{4.13}$$

By $\mathcal{B}(t)$ we indicate the t-th item which has been inserted in set $\mathcal{B}$ according to the insertion order. These inequalities state that at least t overflow items must exist in the range $[\mathcal{B}(t), N]$.

An example. For the sake of clarity we illustrate the CB algorithm with a small example. Consider an OOEBPP instance with 5 items, with size 16, 40, 40, 45 and 50. For each iteration of the CB algorithm, starting from item 5 down to item 1, we report in Table 4.2.2 the considered item, its size, the number of bins currently used, the overall residual capacity R, the set of overflow items $\mathcal{T}$ and the set $\mathcal{B}$.

```
Combinatorial Bounding Algorithm

Input: An ordered set $\mathcal{N}$; a weight $a_i$ for each $i \in \mathcal{N}$; a capacity coefficient $b$.
Output: A set of items $\mathcal{B}$ and a set of overflow items $\mathcal{T}$;
        a lower bound on the number of bins $|\mathcal{T}|$

begin
  /* Initialization */
  $\mathcal{B}$:= $\emptyset$; $\mathcal{T}$:= $\emptyset$; $R$:= 0;
  $S$:= $\emptyset$ /* the set of candidate overflow items */

  for $i$:= $|\mathcal{N}|$ down to 1 do
    $S$:= $S \cup \{i\}$
    if $a_i > R$ then
      /* Choose the candidate of maximum weight */
      $j^*$:= $\text{argmax}_{j \in S}\{a_j\}$
      $\mathcal{B}$:= $\mathcal{B} \cup \{i\}$; $\mathcal{T}$:= $\mathcal{T} \cup \{j^*\}$; $S$:= $S \setminus \{j^*\}$
      $R$:= $R + (b-1) + a_{j^*}$
    $R$:= $R - a_i$
  /* Output */
  return $\mathcal{B}$, $\mathcal{T}$ and $|\mathcal{T}|$
end
```

Figure 4.1: Computation of the combinatorial lower bound.

Item	Size	bins	R	$\mathcal{T}$	$\mathcal{B}$
5	50	1	49	$\{5\}$	$\{5\}$
4	45	1	4	$\{5\}$	$\{5\}$
3	40	2	58	$\{5,4\}$	$\{5,3\}$
2	40	2	18	$\{5,4\}$	$\{5,3\}$
1	16	2	2	$\{5,4\}$	$\{5,3\}$

Note that, when item 3 is considered, the residual capacity is equal to 4 and it is not enough to accommodate the item. Hence a second bin is initialized: its overflow item is item 4, while the current item, that is item 3, is inserted into the set $\mathcal{B}$. The final solution uses only two bins and it is not feasible. In this example we have $\mathcal{B}(1) = 5$ and $\mathcal{B}(2) = 3$ and we can add to the master problem the two inequalities:

$$\sum_{k \in K_5} z_k \geq 1 \quad \text{and} \quad \sum_{k \in K_3 \cup K_4 \cup K_5} z_k \geq 2.$$

4.3 The pricing problem

The sets K_j of feasible columns have exponentially many elements; therefore a restricted set covering problem (RSCP) is considered, and additional columns with negative reduced cost are iteratively generated by need, with a column generation algorithm. For each given $j \in \mathcal{N}$ the pricing problem we need to solve to generate a new column is a binary knapsack problem. Thus a negative reduced cost column can be generated by solving at most $|\mathcal{N}|$ binary knapsack problems. However solving a large number of knapsack problems to optimality to generate negative reduced cost columns can be unnecessary, since we just need one negative reduced cost column, provided it exists. Therefore we solve a pricing problem in which the overflow item is not fixed, but rather it must be chosen in an optimal way, that is we search for the column of minimum reduced cost for all possible choices of the overflow item. The pricing problem is the following:

$$\begin{aligned}
\text{minimize } \pi(\lambda,\mu) = & 1 - \sum_{i\in\mathcal{N}} \lambda_i (y_i + \sum_{j>i} x_{ij}) - \sum_{i\in\mathcal{N}} y_i \sum_{t|\mathcal{B}(t)\le i} \mu_t \\
\text{s.t.} & \sum_{i<j} a_i x_{ij} \le (b-1) y_j && \forall j \in \mathcal{N} \\
& \sum_{i\in\mathcal{N}} y_i = 1 \\
& y_i \in \{0,1\} && \forall i \in \mathcal{N}, \\
& x_{ij} \in \{0,1\} && \forall j \in \mathcal{N},\ \forall i < j
\end{aligned}$$

Coefficients λ_i are the non-negative dual variables associated to covering constraints (4.11) and coefficients μ_t are the non-negative dual variables associated to valid inequalities (4.13).

After defining $\rho_i = \lambda_i + \sum_{t|\mathcal{B}(t)\le i} \mu_t$, the pricing problem can be rewritten in an equivalent way as follows:

$$\begin{aligned}
\text{minimize } \pi(\lambda,\mu) = & 1 - \sum_{i\in\mathcal{N}} (\lambda_i x_i + \rho_i y_i) && && (4.14) \\
\text{s.t.} & \sum_{i\in\mathcal{N}} a_i x_i \le b-1 && && (4.15) \\
& \sum_{i\in\mathcal{N}} y_i = 1 && && (4.16) \\
& x_i + \sum_{j\le i} y_j \le 1 && \forall i \in \mathcal{N} && (4.17) \\
& x_i, y_i \in \{0,1\} && \forall i \in \mathcal{N}
\end{aligned}$$

In this model each binary variable y_i is equal to 1 if and only if item i is assigned to the bin and it is the overflow item, while each binary variable x_i is equal to 1 if and only if item i is assigned to the bin and it is not the overflow item. The capacity constraint (4.15) only concerns the x variables. Constraints (4.17) impose the given ordering to the items: if item i is assigned to the bin and is not the overflow item, then no item j with $j \leq i$ can be the overflow item. Constraint (4.16) states that there must be exactly one overflow item in the bin; it is implied by constraints (4.17) whenever, as in our case, $\rho_i \geq \lambda_i \ \forall i \in \mathcal{N}$.

We call this subproblem the ordered open-end knapsack problem (OOEKP). For analogy with the binary knapsack problem, we state the objective function (4.14) in maximization form as follows:

$$\pi(\lambda, \mu) = 1 - \max \ \{\sum_{i \in \mathcal{N}} (\lambda_i x_i + \rho_i y_i)\}$$

In the next section we discuss the exact optimization of the OOEKP.

4.4 A pricing algorithm

For each choice of the overflow item a generic instance of the OOEKP reduces to an instance of the binary knapsack problem (KP), that is a well studied problem and can be effectively solved by a number of existing algorithms [74] [55].

The relation with the KP can be exploited even further: in principle the OOEKP can be solved in $O(|\mathcal{N}|b)$ computing time using standard recursion: for $i = 1 \dots |\mathcal{N}|$ and $w = 0 \dots b-1$ compute

$$f_{i,w} = \begin{cases} f_{i-1,w} & \text{if } w < a_i \\ \max \ \{f_{i-1,w}, f_{i-1,w-a_i} + \lambda_i\} & \text{otherwise} \end{cases}$$

where $f_{0,w} := 0$ for all $w = 0 \dots (b-1)$. Then, an OOEKP optimum can be found as

$$\pi(\lambda, \mu) = 1 - \max \ _{i \in \mathcal{N}}\{\rho_i + \{f_{i-1,b-1}\}\}$$

and the corresponding solution can be reconstructed by simply keeping a set of pointers during the computation of the $f_{i,w}$ values.

However, this approach is often impractical: more sophisticated and effective techniques have been devised for the KP.

The algorithm we present here performs an implicit search to identify the optimal overflow item, that is the overflow item of an optimal solution. We propose fast bounding and problem-reduction procedures, coupling them with effective algorithms for the KP. Nevertheless, the worst case time complexity of our procedure is worse than that of the dynamic programming approach, since it requires the optimization of a number of KPs that is bounded by N. However, we experimentally observed that the number of KPs to be optimized if often very small, and the computing time of our approach is in practice one order of magnitude better with respect to the dynamic programming procedure.

General description. The algorithm initializes a best incumbent lower bound z_{LB} and a set of candidate overflow items S. Then the algorithm computes upper bounds to the value of the OOEKP for each possible choice of the overflow item. These upper bounds are used both to guide the search in a best-first-search fashion and to terminate the algorithm. After that the algorithm iteratively selects a "most promising" overflow item according to its associated upper bound, it solves a corresponding binary knapsack problem instance and this yields a feasible OOEKP solution. The information provided by the optimal solution of the binary knapsack instance is also exploited by additional fathoming rules to reduce the number of possible candidate overflow items to be considered.

Preprocessing and initialization. Consider the range $\{1, \dots, l\}$ such that $\sum_{j=1}^{l-1} a_j \leq b$ and $\sum_{j=1}^{l} a_j > b$. The optimal solution of the OOEKP involving only items in $\{1, \dots, l\}$ can be computed in linear time since the capacity constraint is inactive, when the overflow item is not after position l in the given sequence. This optimal value is kept as an initial lower bound z_{LB} and all items in the range $\{1, \dots, l\}$ are no longer considered as candidate overflow items.

Reduction. Some more items that cannot be optimal overflow items are identified as follows. For each pair of items i and j with $i < j$ such that $\rho_i \leq \rho_j$, item i can be discarded from the set S of candidate overflow items: given a feasible OOEKP solution with i as the overflow item, a non-worse feasible OOEKP solution can be obtained by simply replacing item i with item j, since feasibility is not affected by the size of the overflow item and the objective function value does not decrease.

Notation. In the remainder we use the following notation. With KP_j we indicate the optimal value of the binary knapsack problem instance in which the only items available are those in the range $[1, \dots, j-1]$, with the capacity of the knapsack equal to $b - 1$.

$$KP_j = \max \ \{\sum_{i=1}^{j-1} \lambda_i x_i : \sum_{i=1}^{j-1} a_i x_i \leq b - 1, x_i \in \{0, 1\} \ \forall i = 1, \dots, j-1\}$$

We indicate by LKP_j the optimal solution of the linear relaxation of KP_j:

$$LKP_j = \max \ \{\sum_{i=1}^{j-1} \lambda_i x_i : \sum_{i=1}^{j-1} a_i x_i \leq b - 1, 0 \leq x_i \leq 1 \ \forall i = 1, \dots, j-1\}$$

Obviously

$$LKP_j \geq KP_j. \tag{4.18}$$

With $OOEKP_j$ we indicate the optimal value of the ordered open-end knapsack problem in which item j has been selected to be the overflow item.

$$OOEKP_j = KP_j + \rho_j \tag{4.19}$$

Step 1: computation of upper bounds. The first step of our algorithm consists in computing an upper bound u_j for each possible choice of the overflow item $j \in S$. For the definitions above, the value

$$u_j = LKP_j + \rho_j \tag{4.20}$$

is an upper bound to the optimal value of the OOEKP in which j is the overflow item:

$$u_j \geq OOEKP_j.$$

The computation of each upper bound u_j requires the optimization of a continuous knapsack problem, that can be carried out in $O(N)$ time [5]. However, instead of solving N continuous knapsack subproblems, the optimal solution of each of them can be obtained by suitably exploiting the structure of the optimal solution of the previous one and this yields a significant reduction in computing time. Consider the *efficiency* of each item j, that is the ratio $e_j = \lambda_j/a_j$ and consider a list T of the items sorted by non-increasing value of efficiency. This is computed in $O(NlogN)$ time. The optimal solution of a continuous knapsack problem can be found by selecting items according to the efficiency order, until an item w is found whose weight a_w exceeds the residual capacity. In order to fill the knapsack such item, called *break item*, is taken with a fractional value. In our algorithm we scan the set of candidate overflow items S, starting from item N down to item l and we scan the ordered list T from the most to the least efficient item; assume LKP_j has been computed and let $i \in S$ be the next candidate overflow item to be considered; assume w is the current break item in the optimal solution of value LKP_j. In the next iteration all items from $j-1$ down to i become unavailable and the corresponding variables are fixed to 0. If some of these variables are basic in the solution of previous continuous knapsack, this yields some slack capacity available in the knapsack, which can be filled by other items, which are chosen scanning T from w onward. When both lists have been sorted in $O(NlogN)$ time, the worst-case computational complexity of the remaining procedure is $O(N)$, because each element of each list is considered only once.

Step 2: Search. In the second step at each iteration the most promising overflow item k is chosen: $k = \text{argmax}_{j \in S}\{u_j\}$ where S is the set of candidate overflow items not yet considered or fathomed. As soon as u_k is found to be not greater than the best incumbent lower bound z_{LB}, the algorithm terminates. Once the most promising item k has been selected, a binary knapsack problem is solved, where the only available items are those with index less than k.

To solve binary knapsack problem instances we used Pisinger's MINKNAP algorithm [88], that is very fast and exploits the optimal solution of the continuous relaxation both as a dual bound and to identify a good starting primal solution. Every time we optimize a binary knapsack problem instance we get an optimal value KP_k: the corresponding solution can be exploited to skip the computation of further binary knapsacks and to obtain feasible OOEKP solutions. Let such solution be defined as

$$\overline{x} = \text{argmax}\{\sum_{i=1}^{k-1} \lambda_i x_i | \sum_{i=1}^{k-1} a_i x_i \leq b - 1, x_i \in \{0,1\} \; \forall i = 1, \ldots, k-1\}$$

and $h = \max \; \{i | x_i = 1\}$ be the first non-zero component in $\overline{x}$. Then, for each $h < j \leq k$, the optimal $OOEKP_j$ solution is given by $\rho_j + KP_k$ and the items in the range $[h+1, \ldots, k]$ can be discarded from the set S; in fact, fixing any $y_j = 1$ with $h < j \leq k$ would not change the optimal solution of the remaining knapsack problem.

The pseudo-code of the pricing algorithm is reported in Figure 4.2.

4.5 Branch-and-price

Branching strategy. Our branching rule is based on the x variables of the compact formulation, since considering the z variables of (4.10)–(4.12) is not effective in this context. Once an optimal solution z^* is obtained, A fractional solution (x^*, y^*) in terms of the original variables can be found exploiting any fractional solution z^* of the reformulated model (and in particular, the optimal one), by fixing $x^*_{ij} = \sum_{j>i} \sum_{k \in K_j} x^k_i z^*_k$, and $y^*_j = \sum_{k \in K_j} z^*_k$ for each $i, j \in \mathcal{N}$.

We have adopted a two-levels branching strategy: in the first level search tree branching decisions are taken on the y_j variables, that is the overflow items of the bins are chosen; the variable with y^*_j value closer to 0.5 is selected and two branches are considered: j is discarded from the set of candidate overflow items in the first branch and fixed as an overflow item in the second branch. In the second level search tree, where the number of bins has been defined and the overflow item of each bin has been chosen, we need to solve a feasibility problem, that resembles the decision version of a generalized assignment problem. In this second level search tree we do a binary branching similar to the previous one, selecting the x_{ij} variable whose value is closest to 0.5.

While the fixing of the x_{ij} variables only reduces the dimension of the subproblems, the pricing algorithm should take into account the fixing of each y_j variable.

Optimization algorithm for the OOEKP

Input: An ordered set $\mathcal{N}$; for each $j \in \mathcal{N}$, a weight a_j, a prize λ_j for the insertion into the knapsack and a prize ρ_j for being the overflow item; a capacity b.
Output: An optimal OOEKP solution (x^*, y^*) and its value z_{LB}

```
begin
  /* Initialization */
  $z_{LB}$:= $-\infty$; $l$:= 1
  while ($\sum_{j=1}^{l-1} a_j \le b-1$) do
    if ($\sum_{j=1}^{l-1} \lambda_j + \rho_l > z_{LB}$) then
      $z_{LB}$:= $\sum_{j=1}^{l-1} \lambda_j + \rho_l$
      $x_j^*$:= 1 $\forall j < l$; $x_j^*$:= 0 $\forall j \ge l$
      $y^*$:= 0; $y_l^*$:= 1
    $l$:= $l+1$
  $S$:= $\{l, \dots, N\}$

  /* Reduction */
  for each $i < j \in S$ do if ($\rho_i \le \rho_j$) then $S$:= $S \setminus \{i\}$

  /* Compute upper bounds from linear relaxations */
  for each $j \in S$ do $u_j$:= $\rho_j + LKP_j$

  /* Examine all candidate overflow items */
  repeat
    /* Select the most promising candidate */
    $k$:= $\text{argmax}_{j \in S}\{u_j\}$
    /* Termination test */
    if ($u_k \le z_{LB}$) then goto end
    /* Solve a KP, store the optimal solution and its value */
    $\overline{x}$:= $\text{argmax}\{\sum_{i=1}^{k-1} \lambda_i x_i : \sum_{i=1}^{k-1} a_i x_i \le b-1, x_i \in \{0,1\}\ \forall i = 1, \dots, k-1\}$
    $KP(k)$:= max $\{\sum_{i=1}^{k-1} \lambda_i x_i : \sum_{i=1}^{k-1} a_i x_i \le b-1, x_i \in \{0,1\}\ \forall i = 1, \dots, k-1\}$
    for each $j = k, \dots, N$ do $\overline{x}_j$:= 0

    /* Identify the best overflow item */
    $h$:= $k$
    while ($\overline{x}_h = 0$) do
      if ($KP(k) + \rho_h > z_{LB}$) then
      /* Update the best incumbent primal solution */
        $z_{LB}$:= $KP(k) + \rho_h$;
        $x^*$:= $\overline{x}$
        $y^*$:= 0; $y_h^*$:= 1
      $S$:= $S \setminus \{h\}$; $h$:= $h-1$
  until ($S = \emptyset$)
end
```

Figure 4.2: Pseudo-code of the OOEKP optimization algorithm

In the first branch ($y_j = 0$) the item j is dropped from the set S of candidate overflow items. In the second branch ($y_j = 1$) two cases must be taken into account: either j is not included in the optimal OOEKP solution, or j is the overflow item; therefore, we first exclude j and solve the remaining OOEKP, then we fix j as overflow item and solve the remaining KP, finally the best of these two solutions

```
Best-Fit Decreasing-Time heuristic

Input: An ordered set $\mathcal{N}$; a weight $a_i$ for each $i \in \mathcal{N}$; a capacity coefficient $b$.
Output: The set of overflow items is a feasible OOEBPP solution $\mathcal{T}$, and the corresponding value $|\mathcal{T}|$


begin
  /* Initialization */
  $\mathcal{T}$:= $\emptyset$; $S$:= $\emptyset$ /* the sets of overflow items and candidate overflow items */
  for $i \in \mathcal{N}$ do $J(i)$:= $\emptyset$ /* each $J(i)$ is the set of items in the bin whose overflow item is $i$ */

  /* BFDT computation */
  for $i$:= $|\mathcal{N}|$ down to 1 do

    $S$:= $S \cup \{i\}$
    /*Compute the set of bins in which $i$ can be inserted */
    $F(i)$:= $\{j \in \mathcal{T} | \sum_{k \in J(j)} a_k + a_i \leq b - 1\}$

    if $F(i) = \emptyset$ then
      /* Initialize a new bin with the candidate of highest weight */
      $i^*$:= $\text{argmax}_{i \in S}\{a_i\}$;
      $\mathcal{T}$:= $\mathcal{T} \cup \{i^*\}$; $S$:= $S \setminus \{i^*\}$
      if $i^* \neq i$ then
        /* Remove $i^*$ from its bin; since $a_i \leq a_{i^*}$ this becomes */
        /* the bin with minimum residual capacity that can host $i$ */
        $j^*$:= $j \in \mathcal{B} | i^* \in J(j)$;
        $J(j^*)$:= $J(j^*) \setminus \{i^*\}$; $J(j^*)$:= $J(j^*) \cup \{i\}$
    else
      $j^*$:= $\text{argmax}_{j \in F(i)}\{\sum_{k \in J(j)} a_k\}$
      $J(j^*)$:= $J(j^*) \cup \{i\}$

  /* Output */
  return $\mathcal{T}$ and $|\mathcal{T}|$
end
```

Figure 4.3: Computation of the Best-Fit Decreasing-Time heuristic.

is taken.

The search tree is explored with a best-bound-first policy.

Primal bounds We have used three different heuristic algorithms to compute primal feasible solutions to the OOEBPP quickly.

The first one is an adaptation of the well-known Best-Fit Decreasing-Height (BFDH) approximation algorithm [67], that we indicate as Best-Fit Decreasing-Time (BFDT) algorithm. The items are iteratively considered from item N down to item 1 and in each iteration the current item is packed into the bin with the minimum residual capacity among those which can accommodate it; if no bin can receive the item, a new bin is initialized. The pseudo-code of this algorithm is reported in Figure 4.3.

A second way of computing feasible solutions is a simple randomized version of the BFDT algorithm, called r-BFDT, in which a set of r items is drawn from a uniform probability distribution and the corresponding bins are initialized. We considered values of the r parameter ranging from 1 to $\lfloor 0.5 \sum_{j\in\mathcal{N}} a_j/b \rfloor$, and we ran r-BFDT 10 times for each value of r.

A third heuristics consists in taking the current fractional solution of the linear relaxation of the master problem and to round up some of the basic z variables, until all rows are covered by columns associated to variables equal to 1. This rounding is carried out in a greedy way, by considering the z_k variables in order of non-increasing fractional value.

The BFDT and r-BFDT heuristics are executed once, as a preprocessing step of our algorithm. The primal solutions obtained in this way are also used to populate the initial RSCP. Instead, the rounding heuristic is used once for each node of the search tree, when the column generation process is over.

Columns deletion and re-insertion. We found useful to periodically remove unpromising columns from the RSCP: each time a node of the search tree is considered, the columns in the RSCP whose reduced costs are higher than a threshold are shifted into a separate pool. The reduced cost of each column is computed with respect to the optimal dual solution on the ancestor node. In our implementation, the removal threshold is computed as $1/(2N)$.

The columns pool is scanned at each column generation iteration: whenever a column is found whose reduced cost is negative with respect to the current dual solution, it is re-inserted in the RSCP. Each column is kept into the pool for up to 6 checks.

Lagrangean bounds. The bound obtained by optimizing the set covering reformulation can also be obtained by solving a Lagrangean dual problem [83], when the set of constraints (4.2) is relaxed:

$$\begin{aligned}
\text{minimize} \sum_{j\in\mathcal{N}} y_j - \sum_{i\in\mathcal{N}} \lambda_i \Big(y_i + \sum_{j>i} x_{ij} - 1\Big) & \\
\text{s.t.} \sum_{i<j} a_i x_{ij} \le (b-1) y_j & \qquad \forall j \in \mathcal{N} \\
x_{ij} \in \{0,1\} & \qquad \forall i < j \in \mathcal{N} \\
y_j \in \{0,1\} & \qquad \forall j \in \mathcal{N}
\end{aligned} \tag{4.21}$$

For each set of multipliers λ the problem decomposes into independent subproblems, one for each $j \in \mathcal{N}$:

$$\begin{aligned}
\text{minimize } & (1-\lambda_j)y_j - \sum_{i<j} \lambda_i x_{ij} \\
\text{s.t.} & \sum_{i<j} a_i x_{ij} \leq (b-1)y_j && \forall j \in \mathcal{N} \\
& x_{ij} \in \{0,1\} && \forall i \leq j \in \mathcal{N} \\
& y_j \in \{0,1\}.
\end{aligned}$$

Each subproblem j can be optimized as follows. First, variable y_j is fixed to 1 and the remaining binary knapsack problem is solved, obtaining a value $\pi_j(\lambda)$. Then, if $\pi_j(\lambda) > 0$, a better solution is found by fixing y_j to 0, and by consequently setting $x_{ij} = 0$ for each $i < j$. Hence, for any choice of the λ_i multipliers, a valid lower bound $\omega(\lambda)$ for OOEBPP is given by:

$$\omega(\lambda) = \sum_{i \in \mathcal{N}} \lambda_i + \sum_{j \in \mathcal{N}} \min \ \{\pi_j(\lambda), 0\}$$

It is worth noting that the main advantage of our pricing method is actually to avoid the computation of such a large number of knapsack problems, since in the OOEKP algorithm we implicitly consider all these π_j values to identify the one with minimum value. However, a lower bound $\bar{\omega}(\lambda)$ on $\omega(\lambda)$ can be obtained by substituting any $\pi_j(\lambda)$ value with a corresponding lower bound $\bar{\pi}_j(\lambda)$. Therefore, we initially set the $\bar{\pi}_j(\lambda)$ values to the u_j bounds, which are readily available after the computation of a OOEKP; then, whenever a binary knapsack problem is solved in order to obtain a KP_j value during the computation of an OOEKP, a corresponding set of $\bar{\pi}_j(\lambda)$ bounds can be updated. Finally, the information drawn from the combinatorial bound can be used to strengthen $\bar{\omega}(\lambda)$: a set of inequalities for the compact formulation analogous to constraints (4.13) is the following:

$$\sum_{i \geq \mathcal{B}(t)} y_i \geq t \qquad \forall t = 1 \ldots |\mathcal{B}|. \tag{4.22}$$

A dual bound $\bar{\omega}(\lambda)$ can be computed as follows. First, the best set of overflow items that satisfy constraint (4.22) is identified; then further items with negative $\bar{\pi}_j(\lambda)$ value are selected. This procedure is detailed in Figure 4.4. Finally, $\bar{\omega}(\lambda)$ is calculated as

$$\bar{\omega}(\lambda) = \sum_{i \in \mathcal{N}} \lambda_i + \sum_{j \in \mathcal{T}} \bar{\pi}_j(\lambda)$$

Whenever, during the column generation iterations, the difference between the highest $\bar{\omega}(\lambda)$ value encountered and the RSCP optimal value is less than 10^{-6}, the

```
Select overflow items:

Input: A set of $\bar{\pi}_j(\lambda)$ values
Output: A set $\mathcal{T}$ of selected overflow items

begin
    $\mathcal{T} = \emptyset$
    for each $t \in \mathcal{B}$ do
        $j^*(t)$:= $\text{argmax}_{j \in \mathcal{N} \setminus \mathcal{T}, j \geq \mathcal{B}(t)} \{\bar{\pi}_j(\lambda)\}$
        $\mathcal{T}$:= $\mathcal{T} \cup \{j^*(t)\}$
    $\mathcal{T}$:= $\mathcal{T} \bigcup \{j \in \mathcal{N} | \bar{\pi}_j(\lambda) < 0\}$
end
```

Figure 4.4: Finding the best valid selection of overflow items

column generation process is terminated, and such Lagrangean bound is kept as a lower bound. Although for each set of multipliers the inequalities (4.22) may help in strengthening the Lagrangean bound, for the optimal choice of multipliers (that is, at the end of the column generation process), this Lagrangean bound never exceeds the value of the set covering LP optimal solution. In fact, once the $\pi_j(\lambda)$ values are computed or approximated, the remaining subproblem can be solved in polynomial time.

Multiple Pricing. The equivalence with Lagrangean relaxation is exploited also to search for different sets of columns at each column generation iteration. In fact, it is a common practice in Lagrangean-relaxation based algorithms to iteratively improve a dual solution with subgradient optimization [47]. Once again, in our case the subgradients are not readily available, since the computation of several binary knapsack problems is avoided. However, the solution corresponding to each LKP_j value, that is computed in the preprocessing step of the OOEKP algorithm, can be used as an approximation of the solutions of the KP_j value, whenever the exact optimization of the KP is not carried out during the OOEKP computation.

At each column generation step, we initialize the λ_i values with the current set of dual variables; we perform at most 50 subgradient iterations, starting with a dumping parameter value of 2.0 and halving it every 10 not improving iterations.

Whenever a set of multipliers is found, that improves the $\bar{\omega}(\lambda)$ bound found in the earlier subgradient iterations, the column corresponding to the OOEKP optimal solution is inserted in the RSCP.

This technique yields substantial improvements in the convergence rate of the column generation algorithm, and helps in avoiding stability problems.

Variable Fixing with Lagrangean Penalties. We also used the $\bar{\pi}_j(\lambda)$ values in a variable fixing procedure. Our aim is to evaluate the effect of complementing the selection of each overflow item.

Therefore, consider each set of items $j \geq \mathcal{B}(t)$ for $t = 1 \dots |\mathcal{B}|$. When the corresponding constraint (4.22) is not active, the effect of complementing can trivially be done by adding or subtracting a $\bar{\pi}_j(\lambda)$ value from the bound. On the opposite, when this constraint is active, the dropping of an overflow item requires the selection of another one in the corresponding interval; in a similar way, the selection of an additional overflow item may allow the dropping of the least profitable selected one.

Hence, for each t, let $\pi^{BO}(t)$ be the minimum $\bar{\pi}_j(\lambda)$ value between the unselected items $j \in \mathcal{N} \backslash \mathcal{T}, j \geq \mathcal{B}(t)$, and $\pi^{WI}(t)$ be the maximum $\bar{\pi}_j(\lambda)$ value between the selected items $j \in \mathcal{T}, j \geq \mathcal{B}(t)$ (BO stands for 'Best Out' and WI stands for 'Worst In'). If $|\{j | j \in \mathcal{T}, j \geq \mathcal{B}(t)\}| > t$, set $\pi^{BO}(t) = \pi^{WI}(t) = 0$, if $\pi^{WI}(t) < 0$ set $\pi^{WI}(t) = 0$.

Then, for each $t = 1, \dots, |\mathcal{T}|$

- for each $j \geq \mathcal{B}(t)$ such that $j \in \mathcal{N} \setminus \mathcal{T}$, if $\lceil \bar{\omega}(\lambda) + \bar{\pi}_j(\lambda) - \pi^{WI}(t) \rceil \geq UB$ then j can be discarded from the set of candidate overflow items ($y_j = 0$),

- for each $j \geq \mathcal{B}(t)$ such that $j \in \mathcal{T}$, if $\lceil \bar{\omega}(\lambda) - \bar{\pi}_j(\lambda) + \pi^{BO}(t) \rceil \geq UB$ then j can be fixed as an overflow item ($y_j = 1$),

since in both cases, the opposite choice would push the lower bound above the upper bound, causing the fathoming of the node.

4.6 Computational results

We tested our branch-and-price algorithm on two datasets proposed in the literature for bi-dimensional packing problems. The first dataset is described in [67] and consists of 5 classes of instances: BENG (10 instances), CGCUT (3 instances), GCUT (4 instances), HT (9 instances) and NGCUT (12 instances). The second dataset is described in [68] and consists of 500 instances, divided into 10 classes of 50 instances, named MV and BW. In bi-dimensional packing problems each item has both a width and a height and the aforementioned datasets contain instances with different types of correlation between these two parameters. In order to obtain OOEBPP instances, we interpreted the height of each item as a "time-stamp": if item i has a smaller height than item j in the bi-dimensional packing instance, then item i precedes item j in the corresponding OOEBPP instance. To obtain a total ordering, we broke the ties according to the order given in the original data file.

Our branch-and-price algorithm was implemented in C++. CPLEX 8.1 was used to solve the LP relaxations. The code was compiled with the GNU CC version 3.2.2, by setting full optimizations. Our computational results were obtained on a Linux workstation equipped with a Pentium IV 1.6GHz processor and 512MB of RAM. We imposed to every test a time limit of one hour.

Dual bounds. In a first set of test, we compared the tightness of the dual bounds proposed in the paper. In tables 4.1(a) and 4.1(b) we report our results on the first and second dataset respectively. Each table is made by six horizontal blocks: in the first one we include the class of instances, while each of the subsequent five blocks refers to the dual bounding technique indicated in the leading row. We denote the combinatorial bound with CB, the linear relaxation of the compact formulation with LP, the relaxation given by the set covering formulation, neglecting constraints (4.13) by CG and the relaxation given by the set covering formulation when the (4.13) inequalities are introduced with MIX. Each entry of the table represents the average value of the instances in a class.

For CB we report the average dual gap, computed as the difference between the optimal value and the value of the bound, divided by the optimal value. We do not report the computation time, because the effort for computing CB and LP for these instances is negligible, and the computation of the other two bounds never required more than a few seconds.

The CG bound is always tight: on the first dataset no duality gap was observed when rounding the value of the CG bound up to the nearest integer; on the second dataset, a gap was found on three classes only, and it was always smaller than 0.2%. The competitor is CB: it is tighter and faster to compute than LP; it gives rather tight bounds (except for class BW06, where the duality gap is more than 11%). In class MV02 it is better than the CG bound too. It is worth noting that combining CG and CB techniques in the MIX relaxation yields sometimes (e.g. on a set of GCUT instances) a bound that is tighter than the best of two.

Primal bounds. In tables 4.2(a) and 4.2(b) we report the cost of the feasible solutions found by three heuristics at the root node, for the instances in the first and second dataset respectively. These tables consist of four blocks: the first one indicates the class of instances, while the subsequent blocks refer to the BFDT, randomized BFDT and rounding heuristics respectively. Each entry indicates the average gap between the value of the heuristic solution and the optimal value, divided by the optimal value. Randomizing the BFDT heuristic yields better primal bounds and allows to obtain a good initial RSCP. The rounding heuristic yielded essential improvements only for instances in the class GCUT of the first dataset.

Optimal solutions. Finally, we performed a set of tests on the effectiveness of branch-and-price for solving the OOEBPP to optimality, comparing it with CPLEX 8.1 used as an IP solver.

A version of the branch-and-price algorithm using the combined bound was able to reduce the duality gap very quickly on all instances; however, the relaxed solutions were highly fractional, and it was hard for a heuristic to find the optimal solution. In the most successful version of our method the inequalities (4.13) were dropped and each μ_t term fixed to 0. Instead, both the CG and CB bounds were computed at each node of the branching tree, and the tightest of them was considered. In this way optimal solutions were found earlier, and less nodes of the branching trees were explored to prove optimality. Therefore we report our computational results only for this last implementation. All the CPLEX parameters were kept at the default values.

The results of our comparison on the first and second datasets are reported in tables 4.3(a) and 4.3(b). In the first column we indicate the instance class name; then, each table has a block for the results of CPLEX and a block for those of branch-and-price. For the first dataset we report the average gap between the value of best solution found and the optimal value, divided by the optimal value ("primal-opt gap"), and the time required to obtain a proven optimal solution ("time"). Both methods complete the computation within the resource limits, but branch-and-price was almost always faster than CPLEX; in particular on classes BENG and GCUT it was two orders of magnitude faster. In Table 4.3(b) related to the second dataset, we indicate also the number of solved instances in each class ("solved instances"). Branch-and-price solved all the instances but 4, while CPLEX failed on 30 instances. CPLEX exceeded the time limit in 16 cases, and had memory overflow problems in the remaining 14; branch-and-price failures were all due to memory overflow. The remaining instances were solved on the average in less than one minute. Hence, it seems that CPLEX failed each time a "hard" instance was encountered, while branch-and-price showed a much more robust behavior. Finally for the classes in which both methods solved all the instances, branch-and-price was always faster and it was effective also on instances that CPLEX failed to optimize.

	CB	LP		CG		MIX	
	dual gap	ceil dual gap	dual gap	ceil dual gap	dual gap	ceil dual gap	dual gap
BENG	0.00%	0.00%	-7.24%	0.00%	-7.23%	0.00%	0.00%
CGCUT	0.00%	0.00%	-6.26%	0.00%	-3.50%	0.00%	0.00%
GCUT	-5.00%	-5.00%	-11.51%	0.00%	-3.33%	0.00%	-2.50%
HT	0.00%	-2.78%	-11.40%	0.00%	-11.05%	0.00%	0.00%
NGCUT	-4.71%	-6.10%	-20.97%	0.00%	-11.30%	0.00%	0.00%
Average	-2.01%	-3.11%	-12.93%	0.00%	-8.72%	0.00%	-0.26%

(a)

	CB	LP		CG		MIX	
	dual gap	ceil dual gap	dual gap	ceil dual gap	dual gap	ceil dual gap	dual gap
MV01	-2.95%	-2.95%	-5.69%	0.00%	-2.78%	0.00%	-1.16%
MV02	0.00%	-0.13%	-7.13%	-0.13%	-7.02%	0.00%	0.00%
MV03	-1.24%	-1.40%	-4.98%	-0.11%	-3.05%	-0.11%	-0.79%
MV04	0.00%	0.00%	-6.68%	0.00%	-6.46%	0.00%	0.00%
BW01	-1.39%	-1.56%	-5.56%	-0.17%	-3.00%	-0.17%	-1.00%
BW02	0.00%	0.00%	-7.22%	0.00%	-7.02%	0.00%	0.00%
BW03	-1.72%	-2.42%	-7.41%	0.00%	-4.99%	0.00%	-1.47%
BW04	-8.99%	-9.37%	-11.98%	0.00%	-1.07%	0.00%	-1.04%
BW05	-11.72%	-11.78%	-14.89%	0.00%	-1.72%	0.00%	-1.72%
BW06	-0.18%	-0.18%	-5.32%	0.00%	-4.80%	0.00%	-0.17%
Avg.	-2.82%	-2.98%	-7.69%	-0.04%	-4.19%	-0.03%	-0.73%

(b)

Table 4.1: Comparison of dual bounds

	BFDT opt. gap	r-BFDT opt. gap	CG opt. gap
BENG	0.00%	0.00%	0.00%
CGCUT	5.88%	1.96%	1.96%
GCUT	8.33%	8.33%	1.67%
HT	0.00%	0.00%	0.00%
NGCUT	0.00%	0.00%	0.00%
Average	1.34%	1.03%	0.33%

(a)

	BFDT primal gap	r-BFDT primal gap	CG primal gap
MV01	2.59%	1.96%	1.68%
MV02	1.07%	0.89%	0.89%
MV03	6.88%	5.90%	5.57%
MV04	4.28%	4.01%	4.01%
BW01	7.93%	6.58%	5.34%
BW02	2.61%	1.61%	1.61%
BW03	5.06%	3.67%	3.67%
BW04	5.84%	1.95%	1.86%
BW05	6.63%	2.04%	1.55%
BW06	8.10%	6.11%	6.11%
Avg.	5.10%	3.47%	3.23%

(b)

Table 4.2: Comparison of primal bounds

	CPLEX		branch-and-price	
	primal-opt gap	time(s)	primal-opt gap	time (s)
BENG	0.00%	1.73	0.00%	0.08
CGCUT	0.00%	0.09	0.00%	0.04
GCUT	0.00%	100.85	0.00%	1.65
HT	0.00%	0.04	0.00%	0.02
NGCUT	0.00%	0.02	0.00%	0.03
Avg	0.00%	11.09	0.00%	0.21

(a)

	CPLEX			branch-and-price		
	solved instances	primal-opt gap	time (s)	solved instances	primal-opt gap	time (s)
MV01	47	0.00%	16.75	50	0.00%	2.67
MV02	50	0.00%	0.50	50	0.00%	0.02
MV03	45	0.32%	14.73	50	0.00%	20.01
MV04	50	0.00%	11.53	50	0.00%	9.83
BW01	41	0.75%	45.99	50	0.00%	5.49
BW02	47	0.49%	0.97	50	0.00%	14.05
BW03	48	0.19%	38.91	49	0.08%	50.91
BW04	48	0.00%	7.30	50	0.00%	1.37
BW05	48	0.00%	1.98	50	0.00%	0.42
BW06	46	0.42%	46.09	47	0.48%	29.77
Avg.	470	0.22%	18.47	496	0.06%	13.45

(b)

Table 4.3: Solving the OOEBPP to proven optimality

Part II

Assignment Problems

In the next chapter we consider the multilevel generalized assignment problem, a variation of the generalized assignment problem in which a set of tasks has to be assigned to a set of agents, that can work at different efficiency levels. In the general framework of partitioning problems, this can be considered the opposite of packing problems: there are no fixed costs for activating each agent, instead an allocation cost incurs each time a task is assigned to an agent.

In this case, the set of tasks has no special structure. However, the particular composition of allocation patterns can be exploited in the column generation routine: the pricing problem can be solved as a multiple-choice knapsack problem, for which a number of very effective codes exist.

Instead, the main challenge in this problem is the design of effective primal heuristics and branching rules.

Chapter 5

A branch-and-price algorithm for the multilevel generalized assignment problem

The multilevel generalized assignment problem (MGAP) is a variation of the generalized assignment problem, in which agents can execute tasks at different efficiency levels with different costs. We present a branch-and-price algorithm that is the first exact algorithm for the MGAP. It is based on a decomposition into a master problem with set partitioning constraints and a pricing subproblem that is a multiple choice knapsack problem. We report on our computational experience with randomly generated instances with different numbers of agents, tasks and levels and with different correlations between cost and resource consumption for each agent-task-level assignment. Experimental results show that our algorithm is able to solve instances larger than those of the maximum size considered in the literature to proven optimality.

5.1 Introduction

The multilevel generalized assignment problem (MGAP) is a variation of the well-known generalized assignment problem (GAP). The GAP consists of assigning tasks to agents with limited capacity, so that each task is assigned to an agent and a capacity constraint is satisfied for each agent. In the MGAP each task-agent assignment can be made at different levels, implying both different costs (or revenues) and different amounts of resource used.

The MGAP arises in the context of large manufacturing systems: it was first described in [44] as a task allocation problem in a real manufacturing environment.

The problem arises when machines performing manufacturing operations on jobs can work at different "levels": this means that the same job can be executed, for instance, with more or less accuracy, in more or less time or with a larger or smaller energy consumption. Obviously the outcome in terms of product quality or added value also depends on the level on which the manufacturing operations have been done. Levels may also represent different lot sizes as in the original paper by Glover et al.. Besides its application in production planning contexts, due to its combinatorial structure the MGAP can also appear as a subproblem in other contexts such as load balancing in clusters for high performance computing, multi-facility location and multi-vehicle routing problems. For this reason we prefer here the general terms "task" and "agent" instead of "job" and "machine", that are more specific to production scheduling optimization. Since it is a generalization of the GAP, the MGAP is $\mathcal{NP}$-hard and even the problem of determining whether a feasible solution exists is $\mathcal{NP}$-complete.

[64] proposed a tabu search algorithm for the MGAP. They reported on results obtained with instances involving up to 40 tasks, 4 agents and 4 efficiency levels. More recently [39] presented two heuristic algorithms, tested on larger instances with up to 200 tasks, 30 agents and 5 efficiency levels. No ad hoc algorithm has been presented so far for the exact optimization of the MGAP. The only attempts to obtain optimal solutions have been made with general purpose optimization packages, but the very large number of binary variables allows to solve only problem instances of small size. [85] proposed to add logic cuts to strengthen the initial formulation; in this way they could solve problem instances with up to 60 tasks, 30 agents and 2 levels to optimality using CPLEX.

Branch-and-price is an effective mathematical programming technique to solve optimization problems like the GAP and the MGAP, requiring the partition of a set of elements into constrained subsets. A branch-and-price algorithm for the GAP was presented by [95].

In this paper we present a branch-and-price algorithm based on a decomposition of the MGAP into a master problem and a pricing subproblem; the former is a set partitioning problem, while the latter is a multiple choice knapsack problem. We illustrate a branching strategy that is both effective at improving the dual bound and compatible with the combinatorial structure of the pricing subproblem. Our algorithm could solve problem instances larger than those of the maximum size considered in the literature (400 tasks, 80 agents, 4 levels). We compared the branch-and-price algorithm with CPLEX with and without logic cuts, solving random instances with different correlations between cost coefficients and resource requirements.

The paper is organized as follows: in Section 5.2 we introduce the basic formulation of the MGAP and its set partitioning reformulation; in Section 5.3 we describe the branch-and-price algorithm and we discuss some implementation details; in Section 5.4 we present our experimental results and we draw some conclusions.

5.2 Formulations

Consider a set of agents $\mathcal{N} = \{1 \dots N\}$ and a set of tasks $\mathcal{M} = \{1 \dots M\}$, such that each task must be assigned to an agent. Each agent $i \in \mathcal{N}$ can execute each task $j \in \mathcal{M}$ at different efficiency levels $k \in \mathcal{K} = \{1 \dots K\}$. Following [64] we formulate the MGAP as a minimization problem.

$$\begin{aligned}
\min \quad & \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} \sum_{k \in \mathcal{K}} c_{ijk} x_{ijk} & & (5.1)\\
\text{s.t.} \quad & \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} x_{ijk} = 1 & \forall j \in \mathcal{M} \quad & (5.2)\\
& \sum_{j \in \mathcal{M}} \sum_{k \in \mathcal{K}} a_{ijk} x_{ijk} \le b_i & \forall i \in \mathcal{N} \quad & (5.3)\\
& x_{ijk} \in \{0,1\} & \forall i \in \mathcal{N},\ \forall j \in \mathcal{M},\ \forall k \in \mathcal{K} \quad & (5.4)
\end{aligned}$$

This model will be indicated as the 'natural formulation' of the MGAP. Binary variables x are assignment variables: $x_{ijk} = 1$ if and only if task $j \in \mathcal{M}$ is assigned to agent $i \in \mathcal{N}$ at level $k \in \mathcal{K}$. Each task $j \in \mathcal{M}$ implies a resource consumption $a_{ijk} \ge 0$ when it is assigned to agent $i \in \mathcal{N}$ at level $k \in \mathcal{K}$; each agent $i \in \mathcal{N}$ has an amount b_i of available resource. Each agent-task-level assignment implies a cost $c_{ijk} \ge 0$. Set partitioning constraints (5.2) impose that each task is assigned to one agent at one efficiency level. Capacity constraints (5.3) impose the resource restriction for each agent. The objective is to minimize the sum of allocation costs.

Usually the following assumption holds in real cases: for each agent-task pair (i, j) and for each two different levels k and h with $k < h$, we assume $a_{ijk} < a_{ijh}$ and $c_{ijk} > c_{ijh}$. When this property does not hold, some assignments are dominated and the corresponding variables can be fixed to 0 by a trivial preprocessing. It is clear that the correlation between the coefficients plays an important role in making an instance easy or hard to solve. This is explained in more detail in Section 5.4.

Consider the relaxation in which constraints (5.2) are replaced by

$$\sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} x_{ijk} \ge 1 \quad \forall j \in \mathcal{M} \qquad (5.5)$$

Any feasible solution of (5.1) (5.5) (5.3) (5.4) in which some task is assigned more than once can be transformed into a feasible solution of (5.1) (5.2) (5.3) (5.4) by simply deleting the assignments in excess and this does not increase the value of the objective function. Therefore this relaxation has the same optimal value of the natural formulation.

We introduce here an alternative formulation of the MGAP, which is viable for a branch-and-price approach. Let a *duty* d for agent i be an assignment of tasks to agent i, that is a vector $\boldsymbol{x_i^d} = (x_{i11}^d, \ldots, x_{iMK}^d)$, where each component x_{ijk}^d takes value 1 if task j is assigned to agent i at efficiency level k, and 0 otherwise. Let $\mathcal{D}_i = \{x_i^1, \ldots, x_i^{D_i}\}$ be the set of all feasible duties for agent $i \in \mathcal{N}$, i.e. the set of vectors $(x_{i11}^d, \ldots, x_{iMK}^d)$ such that

$$\sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}} a_{ijk}x_{ijk}^d \leq b_i$$

$$\sum_{k\in\mathcal{K}} x_{ijk}^d \leq 1 \qquad \forall j \in \mathcal{M}$$

$$x_{ijk}^d \in \{0,1\} \qquad \forall j \in \mathcal{M} \;\; \forall k \in \mathcal{K}$$

Let z_i^d be a binary variable indicating whether a duty $d \in \mathcal{D}_i$ is selected for agent $i \in \mathcal{N}$. The MGAP can be reformulated as follows.

$$\min \sum_{i\in\mathcal{N}}\sum_{d\in\mathcal{D}_i}\Big(\sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}} c_{ijk}x_{ijk}^d\Big)\, z_i^d \tag{5.6}$$

$$\text{s.t.} \sum_{i\in\mathcal{N}}\sum_{d\in\mathcal{D}_i}\Big(\sum_{k\in\mathcal{K}} x_{ijk}^d\Big)\, z_i^d = 1 \qquad \forall j \in \mathcal{M} \tag{5.7}$$

$$\sum_{d\in\mathcal{D}_i} z_i^d = 1 \qquad \forall i \in \mathcal{N} \tag{5.8}$$

$$z_i^d \in \{0,1\} \qquad \forall i \in \mathcal{N} \;\; \forall d \in \mathcal{D}_i \tag{5.9}$$

We remark that each x_{ijk}^d term represents a constant in this model, therefore expressions (5.6) and (5.7) are linear. In this master problem (MP for short) constraints (5.7) guarantee that each task is assigned to one agent and constraints (5.8) guarantee that one duty is selected for each agent. Both of them can be replaced by inequalities. Partitioning constraints (5.7) can be relaxed into covering constraints

$$\sum_{i\in\mathcal{N}}\sum_{d\in\mathcal{D}_i}\Big(\sum_{k\in\mathcal{K}} x_{ijk}^d\Big)\, z_i^d \geq 1 \quad \forall j \in \mathcal{M} \tag{5.10}$$

for the same reason outlined above. Since the empty duty (i.e. a duty with no assignments) is always feasible for each agent, we can replace constraints (5.8) with

$$\sum_{d\in\mathcal{D}_i} z_i^d \leq 1 \quad \forall i \in \mathcal{N} \tag{5.11}$$

We consider a master problem with inequality constraints (5.10) and (5.11) instead of (5.7) and (5.8) because this makes easier for the simplex algorithm to find feasible solutions when solving its linear relaxation.

In general each set $\mathcal{D}_i$ includes an exponential number of assignments and therefore the master problem has an exponential number of variables. We solve the linear relaxation of the master problem (LMP for short) by column generation: we consider a restricted linear master problem (R-LMP) including only some subsets $\mathcal{D}'_i$ of columns, that is

$$\begin{aligned}
\min\ & \sum_{i\in\mathcal{N}}\sum_{d\in\mathcal{D}'_i}\Big(\sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}} c_{ijk}x^d_{ijk}\Big)\ z^d_i && && (5.12)\\
\text{s.t.}\ & \sum_{i\in\mathcal{N}}\sum_{d\in\mathcal{D}'_i}\Big(\sum_{k\in\mathcal{K}} x^d_{ijk}\Big)\ z^d_i \geq 1 && \forall j\in\mathcal{M} && (5.13)\\
& -\sum_{d\in\mathcal{D}'_i} z^d_i \geq -1 && \forall i\in\mathcal{N} && (5.14)\\
& z^d_i \geq 0 && \forall i\in\mathcal{N}\ \ \forall d\in\mathcal{D}'_i && (5.15)
\end{aligned}$$

where constraints $z^d_i \leq 1$ have been removed since they are implied by constraints (5.14).

Let $\lambda \in R^M_+$ and $\mu \in R^N_+$ be the vectors of non-negative dual variables corresponding to constraints (5.13) and (5.14) respectively. The reduced cost of duty d for agent i is

$$\bar{r}^d_i = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}} c_{ijk}x^d_{ijk} - \sum_{j\in\mathcal{M}} \lambda_j\ \Big(\sum_{k\in\mathcal{K}} x^d_{ijk}\Big) + \mu_i$$

To find columns with negative reduced cost we must solve the following pricing problem for each agent $i \in \mathcal{N}$:

$$\begin{aligned}
\min\ & \bar{r}^d_i = \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}}(c_{ijk}-\lambda_j)x^d_{ijk} + \mu_i && && (5.16)\\
\text{s.t.}\ & \sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}} a_{ijk}x^d_{ijk} \leq b_i && && (5.17)\\
& \sum_{k\in\mathcal{K}} x^d_{ijk} \leq 1 && \forall j\in\mathcal{M} && (5.18)\\
& x^d_{ijk} \in \{0,1\} && \forall j\in\mathcal{M},\ \ \forall k\in\mathcal{K} && (5.19)
\end{aligned}$$

that is a multiple choice knapsack problem (MCKP). Though being $\mathcal{NP}$-hard, the MCKP is well-solvable in practice [74] [55] and this is a reason that makes our column generation approach to the MGAP particularly appealing.

The main computational advantage of the reformulation presented above is that the bound given by the linear relaxation of model (5.6) (5.10) (5.11) (5.9) dominates that given by the linear relaxation of model (5.1) (5.5) (5.3) (5.4). This is due to the convexification of constraints

$$\begin{aligned} &\sum_{j\in\mathcal{M}}\sum_{k\in\mathcal{K}} a_{ijk}x^d_{ijk} \leq b_i \\ &\sum_{k\in\mathcal{K}} x^d_{ijk} \leq 1 \qquad\qquad \forall j \in \mathcal{M}. \end{aligned}$$

The MCKP polyhedra defined by these constraints do not possess the integrality property, since the MCKP is NP-hard. Therefore their convexification yields a lower bound which is guaranteed to be greater than or equal to the linear programming lower bound [76]. In our experiments (see Table 5.1, Section 5.4), the lower bound provided by the reformulation used in our branch-and-price algorithm was actually tighter than the linear programming bound of the original model, which is used by general-purpose MIP solvers.

5.3 A branch-and-price algorithm

5.3.1 Lower bound and termination

We exploit the equivalence between Lagrangean relaxation and Dantzig-Wolfe decomposition in the column generation termination test: the terms μ_i in the pricing problem (5.16)–(5.19) are not relevant in the definition of the optimal solution; hence at each iteration t of column generation the current values of the dual variables $\boldsymbol{\lambda}^t$ are used as multipliers to compute a valid lower bound:

$$\omega^t = -\sum_{i\in\mathcal{N}} \tau^t_i + \sum_{j\in\mathcal{M}} \lambda^t_j$$

where τ^t_i is the optimal value of the pricing subproblem for agent i. In this way a sequence of lower bounds is computed during column generation. This often allowed to prune the current node of the search tree even before column generation was over. When the gap between the optimal value of the R-LMP at iteration t and the best incumbent lower bound is smaller than a predefined threshold, the column generation algorithm is terminated and the best incumbent is kept as the final lower bound. This is useful to avoid undesired tailing-off effects in the column generation algorithm. In our experiments we fixed the threshold to 10^{-6}.

5.3.2 Branching strategy

One of the most challenging aspects in the design of a branch-and-price algorithm is the choice of the branching strategy: besides partitioning the solution space, a good branching strategy must make infeasible the current optimal solution of the R-LMP and it must not change the structure of the pricing subproblem. Many authors have addressed the issue of the design of effective branching strategies in branch-and-bound and branch-and-price algorithms. We refer the reader to [65] and [8] for a detailed treatment of the subject.

We have devised a ternary branching rule, that consists of selecting a task j^* which has a fractional assignment to two or more agents in the optimal solution of the R-LMP: we forbid some of the assignments in each of two new subproblems and we assign a task to a particular agent in the third subproblem.

For each task $j \in \mathcal{M}$ we consider the set M_j of agents for which there is a fractional assignment

$$f_{ij} = \sum_{d \in \mathcal{D}'_i} \sum_{k \in \mathcal{K}} x^d_{ijk} z^d_i$$

in the optimal solution of the R-LMP and we select the agent $i^*_j = \text{argmax}_{i \in \mathcal{N}}\{f_{ij}\}$, corresponding to the highest fractional assignment for task j. The set $M_j \setminus \{i^*_j\}$ is partitioned into two subsets M^-_j and M^+_j in the following way: the agents in $M_j \setminus \{i^*_j\}$ are sorted by non-increasing values of f_{ij} and they are inserted alternately in M^+_j and M^-_j. Agent i^*_j is inserted in both M^+_j and M^-_j. In this way we compute a heuristic solution to a subset sum problem, in order to obtain a balanced partition of the agents in M_j. The idea is to fix the same number of variables in each branch, while trying to keep a balanced partition.

The set of agents to which task j is not assigned, $\widehat{M_j}$, is also partitioned into two subsets $\widehat{M^-_j} = \{i \in \mathcal{N} : f_{ij} = 0, i \leq \tilde{\imath}\}$ and $\widehat{M^+_j} = \{i \in \mathcal{N} : f_{ij} = 0, i > \tilde{\imath}\}$, where $\tilde{\imath}$ is chosen in such a way that $|\widehat{M^-_j}| = \lceil |\widehat{M_j}|/2 \rceil$.

The task j^* selected for branching is the one for which $|M_j|$ is maximum. In case of ties we select the task for which the partition obtained is most balanced, that is $|\sum_{i \in M^-_j} f_{ij} - \sum_{i \in M^+_j} f_{ij}| / \sum_{i \in M_j} f_{ij}$ is minimum.

Then we branch on j^* by setting

- $\sum_{i \in M^-_{j^*} \cup \widehat{M^-_{j^*}}} \sum_{k \in \mathcal{K}} x_{ij^*k} = 0$ in the first branch,
- $\sum_{i \in M^+_{j^*} \cup \widehat{M^+_{j^*}}} \sum_{k \in \mathcal{K}} x_{ij^*k} = 0$ in the second branch and
- $\sum_{k \in \mathcal{K}} x_{i^*_{j^*} j^* k} = 1$ in the third branch.

The addition of constraints in the first and second branch forbids some assignments but it does not change the structure of the pricing problem. The constraint in the third branch is handled in a similar way: we forbid the assignment of task j^* to all agents but $i^*_{j^*}$ and we state as equality the j^*–th constraint of set (5.18) in the pricing problem for agent i^*_j. This does not change the structure of the pricing problem.

We adopt a mixed search strategy. The third branch is always explored first, in a depth-first search fashion. This allows us to quickly re-optimize the LMP and to search for good primal solutions deep in the search tree. The subproblems in the first and second branch are stored as open nodes. Whenever an integer solution is found, or the dual bound for the subproblem exceeds the value of the best incumbent primal solution, the node with the lowest dual bound is retrieved from the open nodes list, in a best-bound-first search fashion. A set of experiments showed that this branching strategy is more effective than a standard two-branches rule: first, the depth first exploration of the third branch helps in quickly finding tight primal bounds; moreover, the assignment of the task to the most desirable agent is forbidden in both the first two branches, and this helps in tightening the corresponding bounds.

When all the variables in a relaxed solution have integer values, the optimal task-level assignment is computed solving a MCKP for each agent.

5.3.3 Column generation

Pricing algorithm. We solve the binary MCKP to optimality by a modified version of Pisinger's algorithm [87], that combines dynamic programming with bounding and reduction techniques. This algorithm was devised for the MCKP with integer coefficients, while in our pricing subproblems the dual variables (as well as the multipliers in Lagrangean relaxation) can be fractional. Therefore we modified the algorithm in a way similar to that described in [18] and [15], that is by relaxing the bounding tests so that the solution computed by the algorithm may differ from optimality by at most a very small positive value ($n \cdot 10^{-9}$ in our experiments, where n is the number of variables left outside the core). Since the classical formulation of the MCKP has equality constraints, we add a set of N dummy elements, each appearing in a constraint of the set (5.18), corresponding to items with zero resource consumption and zero cost.

Columns management. At each iteration of the column generation algorithm all columns which are generated with a negative reduced cost are inserted into the R-LMP.

Whenever the number of columns exceeds a limit, we remove columns from the R-LMP. According to statistical results (see subsection 5.4.2), this limit was set to 3000 in our experiments. The removal criterion depends on three different tests on the reduced cost of each column: so three types of removable columns are considered.

- A column is *red*, if its reduced cost exceeds the gap between the best incumbent feasible solution and the lowest lower bound among all the open nodes of the search tree. In this case the column cannot belong to an optimal solution of any node of the search tree and therefore it is deleted.
- A column is *yellow*, if its reduced cost exceeds the gap between the current R-LMP value and the Lagrangean lower bound. In this case the column cannot belong to the optimal solution of the current node; the column is deleted from the R-LMP and is stored in a yellow pool P_y.
- A column is *green*, if its reduced cost exceeds the same gap as above divided by N. In this case the column can belong to the optimal solution of the current node; it is removed from the R-LMP and is stored in a green pool P_g.

Since every column is related to a particular agent, each pool is partitioned into N sub-pools. The green pool P_g is scanned before executing the pricing algorithm, also at the node of the search tree in which the deletion has occurred: if any column with negative reduced cost is found, it is inserted into the R-LMP. The columns in the yellow pool P_y are considered for reinsertion only in subsequent nodes of the search tree.

Finally, to avoid an excessive growth of the pools, the columns are erased from the pool when their reduced cost is non-negative for a certain number of consecutive evaluations. This parameter was tuned to a value of 6 in our experiments (see subsection 5.4.2).

Initialization. To guarantee that a feasible solution of the R-LMP exists in each node of the search tree a dummy column is inserted into the initial R-LMP; it corresponds to a duty in which all tasks are executed by a dummy agent with infinite capacity. The cost of such a column is set to a very high value, that is $\sum_{j\in\mathcal{M}} \max_{\ i\in\mathcal{N},k\in\mathcal{K}}\{c_{ijk}\}$. Moreover eleven sets of columns are inserted into the initial R-LMP at the root node, corresponding to primal solutions produced by heuristic algorithms. A detailed description of this initialization is reported in subsection 5.3.4.

In order to obtain a warm start, in each non-root node the R-LMP is initialized with the feasible columns of the most recently solved node plus all columns from the pools that have negative reduced cost when they are evaluated with the optimal dual values of the father node.

5.3.4 Primal bounds

The problem of finding a feasible solution to the MGAP is $\mathcal{NP}$-complete. Nevertheless we search for feasible solutions at every node of the search tree, since the availability of good primal bounds may considerably reduce the overall computing time needed to reach a provably optimal solution.

We devised a fast rounding heuristic, and implemented both heuristic algorithms MGAPH1 and MGAPH2 proposed by [39]. Furthermore, we propose a new local search neighborhood (that we call *SHIFT*) and a modification of the local search technique used for MGAPH2 (that we call *SWAP*). In the following paragraphs we outline these algorithms and two local search techniques. Then we describe how each heuristic is used in the branch-and-price algorithm.

For a formal description of MGAPH1 and MGAPH2 we refer to the original paper, while the complete pseudo-codes of the rounding heuristic and the local search algorithms are reported in the appendix.

Rounding heuristic. Let $f_{ij} = \sum_{d \in \mathcal{D}'_i} (\sum_{k \in \mathcal{K}} x^d_{ijk}) z^d_i$ be the (possibly fractional) assignment of task j to agent i corresponding to the fractional R-LMP solution defined by the z^d_i variables. First, each task is assigned to the agent for which f_{ij} is maximum. Let C_i be the resulting set of tasks assigned to agent i. Second, for every agent i a MCKP with integer coefficients is solved to optimality by the algorithm of Pisinger [87]. If a feasible solution can be found for each agent, a primal bound for MGAP is obtained; otherwise the heuristic fails. We observed that in almost all cases, this method finds a feasible solution.

MGAPH1. This algorithm consists of two steps. First, a super-optimal integer solution is built with a greedy approach. For each task i, the agent-level assignment (i, j, k) with the lowest c_{ijk} is selected, possibly violating some capacity constraint. Then, a local search for feasible solutions is performed, shifting tasks from overloaded agents to agents with enough residual resources. The shift corresponding to the minimum increase in the solution value per resource consumption unit is iteratively selected.

MGAPH2. Consider $g_{ijk} = \sum_{d \in \mathcal{D}'_i} x^d_{ijk} z^d_i$. In a construction step a value $r_j = g_{i'jk'} - g_{i''jk''}$ is computed for each task, where (i', j, k') and (i'', j, k'') are the first and the second agent-level assignment for task j that do not violate capacity constraints with the highest g_{ijk} values. The task with the highest r_j is selected, and the (i', j, k') assignment is made. If, due to capacities, a task is found that cannot be assigned to any agent, the heuristic fails. Otherwise, a local search is performed, considering a neighborhood made of all solutions that can be obtained from the current one by swapping two tasks assigned to different agents.

Local search. First, we propose a *SHIFT* procedure: the neighborhood of the current solution is made of all solutions which can be obtained by shifting a task from an agent to another or from an efficiency level to another. As detailed in the appendix, we consider in turn each job, each agent and each efficiency level on that agent. Whenever an improving move is found, it is immediately performed with a first-improve policy. The cost of this local search step is $O(NMK)$. Second, we modified the pairwise swap neighborhood of [39], obtaining a procedure that we call *SWAP* in the following way. We consider all solutions that can be obtained by swapping two tasks assigned to different agents, or by swapping the efficiency levels of two tasks assigned to the same agent. Only the best improving swap is performed at each iteration.

Our rounding heuristic, coupled with the exploration of the *SHIFT* neighborhood, was run at every column generation step. It yielded good upper bounds even in the earlier iterations with a low computational cost, and this was useful also to drive the column removal routine. MGAPH2 was used once for each node in the search tree, using the optimal R-LMP solution. We modified the local search step of MGAPH2 in the following way: first, we explore the *SHIFT* neighborhood and the first improving shift is made, until no more improving shifts can be found; then we explore the *SWAP* neighborhood and the best improving swap is made, until no more improving swaps can be found. The exploration of the *SHIFT* and the *SWAP* neighborhoods is iterated until no more improving moves can be made. The neighborhood *SWAP* is considered only at the root node. The rounding heuristic and the MGAPH1 algorithm were used in the initialization of the R-LMP: we chose to generate 25 sets of columns using the former, randomly drawing each f_{ij} in the interval $[0, 1)$ (see subsection 5.4.2). The details on how the random values were generated are reported in Section 5.4. Also columns corresponding to infeasible solutions were added to the R-LMP.

5.4 Experimental analysis

5.4.1 Test Instances

We tested the branch-and-price algorithm on three classes of instances. Classes C and D are generated using random generators as described by Martello and Toth for the GAP, and extended to the MGAP, while class E is generated as proposed by [64].

- **Class C:** *uncorrelated resource consumption and cost.*
  - a_{ijk} is taken as a random integer from a uniform distribution in $[5, \dots, 25]$
  - c_{ijk} is taken as a random integer from a uniform distribution in $[1, \dots, 40]$
  - $b_i = 0.8 \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} a_{ijk}/(NK)$
- **Class D:** *strongly correlated resource consumption and cost.*
  - a_{ijk} is taken as a random integer from a uniform distribution in $[1, \dots, 100]$
  - c_{ijk} is taken as a random integer from a uniform distribution in $[101 - a_{ijk}, \dots, 121 - a_{ijk}]$
  - $b_i = 0.8 \sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{K}} a_{ijk}/(NK)$
- **Class E:** *resource consumption and cost correlated through an exponential distribution.*
  - a_{ijk} is randomly generated as $1 - 10 \ln[\text{random}(0,1]]$ rounded to the nearest integer with probability p, $a_{ijk} = \infty$ (that is, the assignment of task j to agent i at level k is forbidden) with probability $1 - p$
  - c_{ijk} is randomly generated as $1000/a_{ijk} - 10 \cdot \text{random}(0,1]$ rounded to the nearest integer
  - $b_i = \max \ \{0.8 \sum\limits_{i \in \mathcal{N}} \sum\limits_{k \in \mathcal{K}} a_{ijk}/(NK), \max_{j,k}\{a_{ijk}\}\}$.

By "random$(0,1]$" we mean a random rational value uniformly drawn in the interval $(0,1]$. Such a random value was generated by drawing a random signed integer and dividing this by the constant value "INT_MAX" (on our machine, 4 bytes are used for signed integers, and "INT_MAX" is set to $2^{31} - 1$).

We generated 215 test instances in the following way. For each class we generated two problem sets, with $M = 100$ and $M = 200$ tasks. For $M = 100$ we considered a number of agents N equal to 10, 20 and 30 and a number of levels K equal to 3, 4 and 5; for $M = 200$ we considered a number of agents N equal to 15 and 30 and a number of levels K equal to 4 and 5. For the instances in class E with $M = 100$ the probability p of allowing an agent-task-level assignment was fixed to 1.0, while for the instances in class E with $M = 200$ the case $p = 0.8$ and the case $p = 1.0$ have been considered. Each combination is reported in the first four columns of the tables reported in this section, and consists of five instances; hence each row of the tables reports the average results for these five instances.

As reported in Section 5.2, dominated assignments can be found by simple preprocessing tests. In fact, about 50% of the binary variables were fixed to 0 for the class C instances, about 11% for class D and about 8% for class E. As expected, the percentage of fixed variables increases as the number of levels K increases.

The branch-and-price algorithm was coded in C++ and compiled under Linux OS with gcc version 2.96 with full optimizations. CPLEX 6.5.3 was used as an LP solver. All tests were run on a PC equipped with an Intel P4 1600MHz CPU and

512MB RAM. Each test was stopped in case of memory overflow or whenever a time-out of two hours was exceeded.

5.4.2 Parameters tuning

As discussed in the previous sections, three parameters affect the computational performance of the algorithm. They are the number of columns generated in order to populate the initial RMP, the threshold on the number of columns in the RMP that triggers the columns removal routine and the number of iterations in which a column is kept in the columns pool. In order to tune these parameters, we considered the CPU time needed to solve the root problem of the instances involving 200 tasks. In figure 5.1 we include three charts, in which different settings for each parameter are compared. In each chart, the values in the abscissae correspond to the different settings of the parameter, and the values in the ordinates are the corresponding computation time. We observed that the computation time did not change significantly for different number of levels. Therefore, in each chart we present 8 series, considering each combination of the correlation type and the number of agents, and reporting the average results on the corresponding instances. First, we tried to initialize the RMP with the dummy column only, or with columns corresponding to 5, 10, 25 and 100 primal solutions. It is worth noting that the worst performance corresponds to the initialization of the RMP with only few columns: when considered as constraints in the dual problem, these columns restrict the solution space leaving the dual variables free to assume misleading values. Setting this parameter to 25 seems to be the most appropriate choice. Second, we tried to set the columns removal threshold to 3000, 4000, 5000, 6000 and 7000. Even if sometimes a setting to 5000 columns would be better (instances of type C, with 15 agents) fixing the threshold to 3000 gives the best average CPU time. Third, we tuned the iterations limit on the columns pool by setting it to 0, which means not managing a pool of removed columns, 3, 6, 12 and 18. The value of 6 was found to be the best. A slight increase in the computation time was observed moving from 0 to 3, which reflects the overhead for managing the pool.

5.4.3 Computational results

In the first set of tests, the quality of the bound given by the LMP relaxation is compared to the bound given by the LP relaxation, that is used by CPLEX and other general purpose solvers. In Table 5.1 we report, for both methods, the average integrality gap (‘int. gap’), that is the gap $(v^* - \omega)/v^*$ between the relaxation value ω and the optimal (or the best-known) integer solution value v^*, and the average number of fractional agent-level assignments for each task ('fract.'), that is the ratio between the number of non-zero variables in an optimal fractional solution and M, the number of tasks.

Both relaxations yield a rather tight bound, but the LMP relaxation clearly dominates the LP relaxation also from an experimental point of view. Nevertheless, the time required to obtain the LMP bound is one order of magnitude (sometimes even two orders) higher than the time required to compute the LP relaxation.

The ratio between the LMP integrality gap and the LP integrality gap increases as the ratio M/N decreases and decreases as K increases. This was expected, since the M/N value represents the average number of tasks assigned to the same agent. As reported by [74], at most two variables can assume fractional values in the linear relaxation of a MCKP problem; these two fractional variables must belong to the same multiple choice constraint (5.18). Hence, when a high number of tasks is assigned to the same agent in a fractional solution, several task-level assignments take an integer value, and the convexification of constraints (5.17)–(5.18) has a minor effect. Second, a high K corresponds to a high number of binary variables in the same multiple choice constraint. This does not affect the number of variables with a fractional value; thus, following the previous argumentation, the convexification has a minor effect. On the opposite, for both relaxations the number of non-zero variables in a fractional solution increases as the ratio M/N decreases. On the average, the LMP solution has more fractional assignments than the LP solution. This is especially evident for instances in class D.

In Table 5.2 we report computational results for our method. The table consists of two horizontal blocks. The first refers to the root node and the second to the nodes of the search tree. We indicate $\bar{v}$ the value of the best primal solution found at the root node, v^* the optimal solution value (or the best-known primal bound, when optimality was not proved), $\bar{\omega}$ the optimal LMP value at the root node and ω^* the dual bound at the end of computation. In the 'root node' block we report the value $(\bar{v} - v^*)/v^*$, that indicates how far the initial primal bound is from optimality ('primal gap'), the value $(\bar{v} - \lceil\bar{\omega}\rceil)/\lceil\bar{\omega}\rceil$, that is the gap between the primal and dual bound at the root node ('p.d. gap'), the number of column generation iterations needed to reach a LMP optimum ('iter.'), the number of generated columns ('cols') and the time spent at the root node. In the 'whole search tree' block we indicate the number of nodes evaluated ('ev. nodes'), the gap between the primal and dual bound at the end of computation $(v^* - \lceil\omega^*\rceil)/\lceil\omega^*\rceil$ ('gap'), the number of instances solved to proven optimality ('opt'), and the overall time spent ('time') in the exploration of the whole search tree (including the root node).

By looking at the rightmost three columns of the root node block, it can be noticed that the number of column generation iterations and the time required to solve the relaxation decrease as the number of agents increases. In fact, the insertion of new columns in the R-LMP is governed by an “all-negative” policy: a new column with negative reduced cost can be found for each agent. A larger number of agents means a larger number of columns added to the R-LMP at each column generation iteration, and thus faster convergence.

By analyzing the computational results for the whole search tree, it is clear that instances in class C can be solved very easily: optimality was proven at the root node for all instances but one. This was expected, since resource consumption and costs are not correlated, and an optimal solution can quickly be found by choosing assignments with low resource consumption and low cost.

All the instances in class E were solved to optimality in a few minutes: often a very tight primal bound was found at the root node, and the gap between primal and dual bounds was closed by exploring a few nodes of the search tree. This shows that our branching rule is effective for this kind of instances.

Effect of symmetries. Instances in class D are indeed the hardest ones. A singular effect was experimentally observed: tight primal and dual bounds were obtained at the root node, but the gap could not be closed after many branching steps. We explain this phenomenon with the following observation. Let the *efficiency* of an assignment (i, j, k) be the ratio $1/a_{ijk}c_{ijk}$. As resource consumption and cost are strongly correlated, several assignments have a similar efficiency. Suppose that a task j is fractionally assigned to agent i in the optimal LMP solution of a node in the search tree, and a branching operation forbids this fractional assignment. Even if agents are not identical, an equivalent solution is likely to exist, in which the fractional part of task j is assigned to a different agent i' at the same efficiency, maybe by shifting to agent i the fractional part of another task j' previously allocated to agent i'. Hence, a fractional optimal solution after branching would be a simple rearrangement of the tasks between the agents. This effect is mitigated in class E instances, since the correlation between resource consumption and cost is not linear.

This further motivates the depth-first search feature of our branching rule: the gap can be closed just by finding the optimal integer solution, that is more likely to be found deep in the search tree than at the root node, as fixing task-agent assignments to 1 has a strong effect in the construction of an integer solution.

Large-scale instances. In order to test our algorithm on a more challenging testbed we considered the set of instances presented by [102]. These instances correspond to GAP problems generated with C, D and E correlation types, in which up to 1600 tasks must be assigned to up to 80 agents. We considered the problems involving the assignment of 400, 900 and 1600 tasks. In order to obtain a set of corresponding MGAP instances, we put the tasks on 2, 3 and 4 levels for the instances with 400, 900 and 1600 tasks respectively, and adjusted the capacity of the agents dividing each value by 2, 3 and 4 correspondingly. We imposed no time limitation to these tests. The results obtained by our branch-and-price on these modified instances are reported in Table 5.3. In the first horizontal block of this table we report the original GAP instance ID and the number of agents, tasks and levels of the corresponding MGAP instance. The second block corresponds to the optimization status at the root node. We indicate how far the primal solution found at the root node is with respect to the best known primal solution, the gap between the primal and dual solutions and the time spent. The third block refers to the overall branching tree and contains the gap between the primal and dual bounds at the end of computation, the number of explored nodes and the time required to obtain a proven optimal solution. The last two columns are marked with a dash when the process ran out of memory. We observed the same behavior of the previous analysis: branch-and-price was able to solve all the instances with correlation type C and E; on the opposite, it failed to solve to proven optimality the instances with correlation type D, even though the gap between primal and dual bounds was very small.

Further testing. As discussed in the previous paragraph, instances in class D remain challenging. While it is computationally easy to find a very tight primal bound, it is hard to identify an optimal integer solution. We also measured the Hamming distance between the primal solution found by our heuristics at the root node and the best primal solution encountered while exploring the search tree. Even if a small improvement is made in the solution value, an average distance higher than 100 and 250 is observed for class D instances with $M = 100$ and $M = 200$ respectively (for class E such distance is about 45 and 65). The symmetry in the LMP optimal solutions suggests the presence of symmetries also in the integer optimal solutions. However, even thought symmetric optima could exist, which are less far apart from the initial solution, it would be still hard to cover such a large distance with standard local search methods.

Finally, we remark that the number of efficiency levels does not affect the performance of our method: branch-and-price was able to solve instances involving up to 30 efficiency levels. Although no problem with a so high number of efficiency levels is addressed in the literature, this kind of instances could arise as a discretization of some nonlinear resource consumption function.

5.4.4 Benchmark algorithms

General purpose solver. As a first term of comparison, we present the behavior of ILOG CPLEX 6.5.3 when used as an IP solver to optimize the MGAP.

CPLEX uses a branch-and-cut approach, automatically generating cliques, cover and GUB-cover inequalities in order to strengthen the LP relaxation. All internal parameters were kept at their default values. These include a relative optimality tolerance of 0.01%. While instances in class C were handled quite easily, CPLEX was able to solve only a small number of instances in class E, and no instance in class D, mainly due to memory overflow problems.

Logic Cuts. Logic cuts are a class of inequalities, analogous to cover cuts, that can be obtained in a preprocessing phase from the capacity constraints (5.3). A similar generation of valid cover cuts in a preprocessing phase has been recently adopted by [81] for the case of the GAP. The use of logic cuts to improve the performances of a general-purpose solver for the MGAP is discussed by [85]. As proposed by the authors, we added all non-dominated 1-cuts to the model of MGAP and used CPLEX to solve the new formulation. The 1-cuts can be generated in linear time with a simple procedure.

According to the computational experience described by [85], all CPLEX parameters were kept at their default values, except for the search policy parameter, whose setting was changed from 'best-bound-first' to 'up-branch-first'.

The introduction of logic cuts yielded a substantial improvement to the performances of CPLEX for small instances: the size of the search tree was strongly reduced, avoiding the memory overflow problems. However, the introduction of these inequalities enlarged considerably the dimension of the problem, and the computation time was substantially increased.

In order to assess the effectiveness of the heuristics and local search methods used for branch-and-price, we tried to incorporate them in the optimization process of CPLEX. We kept the same settings used for branch-and-price: we ran MGAPH2 and considered the SWAP neighborhood only at the root node, while MGAPH1 with the SHIFT neighborhood was used once at each node of the branching tree.

Experimental results. In Table 5.4 we compare the performances of the four methods. The table is divided in four horizontal blocks; each block refers to a solution strategy, which is indicated in the leading row. For each method we report the average time to complete computation (when optimality was proven), the number of instances solved to proven optimality and the average gap between the primal and dual bounds, when computation exceeded resource limitation. As outlined before, even if the introduction of logic cuts solves the memory problems, the convergence of the solver is slow. The embedding of the primal heuristics yielded a further improvement to the performances of CPLEX, but the overall behavior of the method was still the same. This test highlighted that the bounding and branching techniques are of key-importance in the algorithmic success of branch-

and-price. Branch-and-price clearly outperformed the other three methods: this is especially evident for instances in class E, where CPLEX was able to prove the optimality of only 9 of the instances with $M = 100$, the introduction of logic cuts allowed to solve 2 more instances, while our algorithm always provided the optimal solution. Instances in class D are difficult for both CPLEX and our algorithm. However our method allows to obtain a smaller gap between the final upper and lower bounds.

A hard instance. [64] described a hard instance involving 30 tasks, 8 agents and 3 efficiency levels. The best solution found by their tabu search method after 120.7 seconds (on a DECstation 5000 / 120MHz) has a cost equal to 691634. [39] tested their heuristics on this instance, finding solutions of cost equal to 714608 and 703912 within 0.2 and 0.1 seconds (on a HP9000 / 700MHz), with MGAPH1 and MGAPH2 respectively. [85], using CPLEX with logic cuts and allowing a relative optimality tolerance of 1.0%, found a solution of value 690624 in 36 hours of computation (on a machine equipped with a Pentium 100MHz CPU). We started branch-and-price from scratch, that means we did not exploit information about the previously known upper bounds. It solved this instance to proven optimality in about three hours of computation, confirming the value of the optimal solution to be 690624.

5.4.5 Conclusions

In this paper we have described a branch-and-price algorithm for the MGAP. At the best of our knowledge this is the first exact method proposed in the literature for the MGAP. It favorably compares against a state-of-the-art general purpose MIP solver and it was able to prove optimality for a very hard MGAP instance, that had not been solved to proven optimality so far. Its success mainly relies upon the tightness of the set covering reformulation, the existence of an effective algorithm to solve the MCKP and a branching rule that does not affect the combinatorial structure of the pricing subproblem. Following an up-branch-first search strategy, the exact optimization algorithm presented here is also suitable for approximation purposes when very large scale MGAP instances are tackled.

Instances				Linear relaxation		LMP relaxation	
Correlation	N	M	K	int. gap	fract.	int. gap	fract.
C	10	100	3	2.35%	1.10	0.34%	1.23
	10	100	4	2.41%	1.10	0.45%	1.33
	10	100	5	1.82%	1.09	0.39%	1.22
	20	100	3	4.59%	1.19	0.38%	1.31
	20	100	4	3.32%	1.19	0.40%	1.43
	20	100	5	3.10%	1.19	0.40%	1.41
	30	100	3	5.96%	1.29	0.34%	1.41
	30	100	4	3.73%	1.28	0.35%	1.50
	30	100	5	0.36%	1.29	0.00%	1.83
Avg. (C)				3.07%	1.19	0.34%	1.41
D	10	100	3	0.12%	1.10	0.06%	1.73
	10	100	4	0.12%	1.10	0.09%	1.71
	10	100	5	0.06%	1.10	0.06%	2.03
	20	100	3	0.67%	1.20	0.60%	2.45
	20	100	4	0.53%	1.20	0.52%	2.67
	20	100	5	0.43%	1.20	0.43%	3.32
	30	100	3	1.06%	1.30	0.91%	2.81
	30	100	4	0.56%	1.30	0.55%	2.87
	30	100	5	0.50%	1.29	0.50%	3.37
Avg. (D)				0.45%	1.20	0.41%	2.55
E	10	100	3	0.16%	1.10	0.02%	1.43
	10	100	4	0.13%	1.10	0.00%	1.13
	10	100	5	0.11%	1.09	0.00%	1.21
	20	100	3	0.65%	1.19	0.03%	1.71
	20	100	4	0.56%	1.20	0.02%	1.78
	20	100	5	0.63%	1.20	0.03%	1.77
	30	100	3	1.86%	1.29	0.07%	2.32
	30	100	4	1.64%	1.30	0.08%	2.20
	30	100	5	1.53%	1.29	0.07%	2.13
Avg. (E)				0.81%	1.19	0.04%	1.74
C	15	200	4	1.07%	1.08	0.23%	1.25
	15	200	5	1.00%	1.07	0.20%	1.26
	30	200	4	0.19%	1.14	0.09%	2.07
	30	200	5	0.00%	1.15	0.00%	2.74
Avg. (C)				0.57%	1.11	0.13%	1.83
D	15	200	4	0.15%	1.08	0.15%	2.56
	15	200	5	0.11%	1.07	0.11%	3.13
	30	200	4	0.66%	1.15	0.66%	4.13
	30	200	5	0.46%	1.15	0.46%	4.71
Avg. (D)				0.34%	1.11	0.34%	3.63
E	15	200	4	0.08%	1.01	0.01%	1.31
(p = 0.8)	15	200	5	0.07%	1.01	0.00%	1.32
	30	200	4	0.31%	1.01	0.01%	1.60
	30	200	5	0.29%	1.01	0.01%	1.58
Avg. (E, p 0.8)				0.19%	1.01	0.01%	1.45
E	15	200	4	0.07%	1.07	0.01%	1.32
(p = 1.0)	15	200	5	0.06%	1.07	0.01%	1.25
	30	200	4	0.27%	1.14	0.01%	1.42
	30	200	5	0.27%	1.14	0.01%	1.57
Avg. (E, p 1.0)				0.17%	1.11	0.01%	1.39

Table 5.1: Comparison between LP relaxation and LMP relaxation

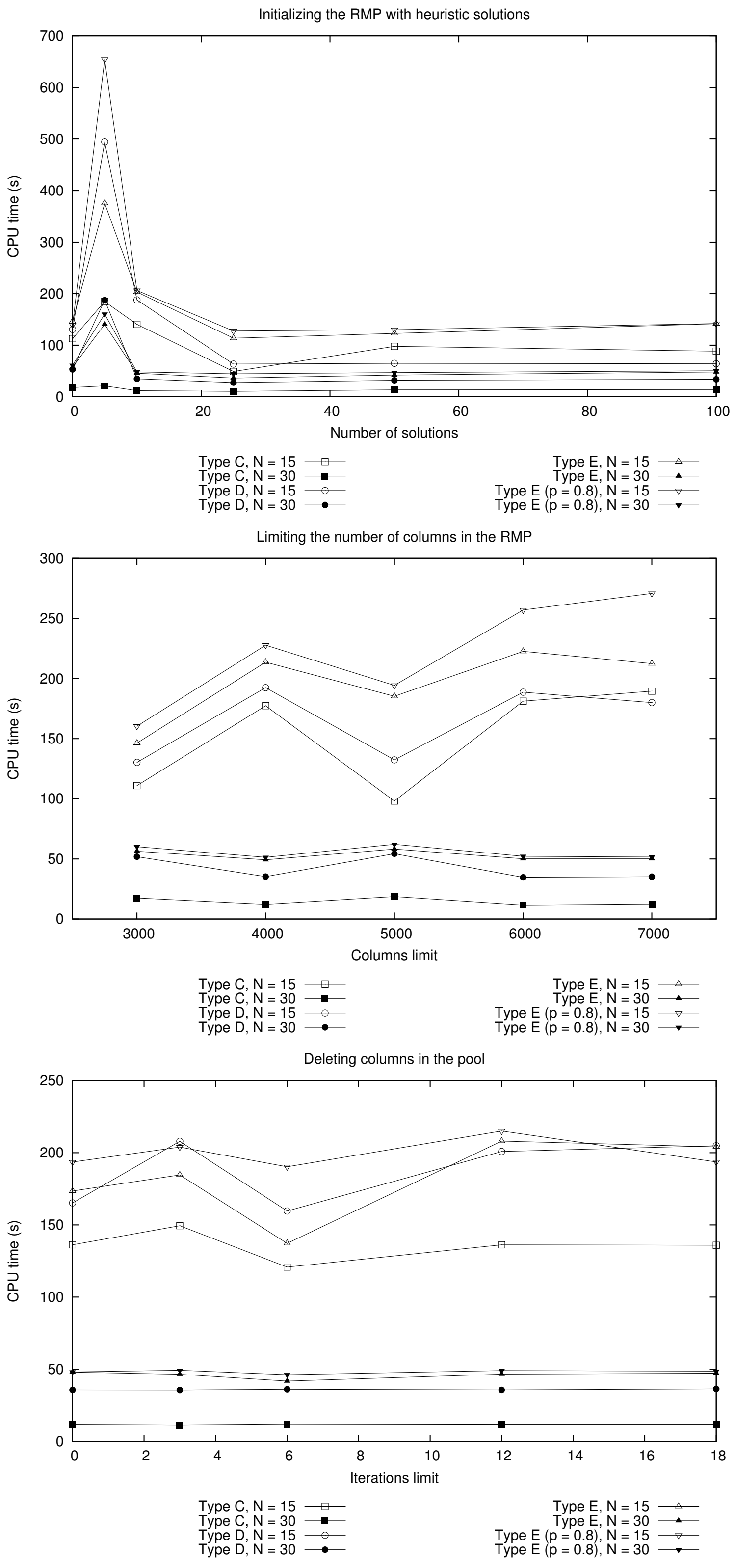


Figure 5.1: Tuning the algorithm parameters

Instances				root node					whole search tree			
Corr.	N	M	K	primal gap	p.d. gap	iter.	cols	time(s)	ev. nodes	gap	opt	time(s)
C	10	100	3	0.00%	0.00%	166.20	1702.40	5.19	0.00	0.00%	5	5.19
	10	100	4	0.00%	0.00%	152.80	1684.40	5.14	0.00	0.00%	5	5.14
	10	100	5	0.00%	0.00%	150.00	1619.40	4.77	0.00	0.00%	5	4.77
	20	100	3	0.00%	0.00%	49.20	1315.20	1.62	0.00	0.00%	5	1.62
	20	100	4	0.00%	0.00%	42.40	1222.40	1.58	0.00	0.00%	5	1.58
	20	100	5	0.00%	0.00%	42.40	1196.20	1.61	0.00	0.00%	5	1.61
	30	100	3	0.00%	0.00%	25.20	1329.00	1.18	0.00	0.00%	5	1.18
	30	100	4	0.98%	0.16%	21.20	1242.60	1.26	1.20	0.00%	5	1.60
	30	100	5	0.00%	0.00%	14.80	1142.80	1.28	0.00	0.00%	5	1.28
Avg. (C)				0.11%	0.02%	73.80	1383.82	2.62	0.13	0.00%	45	2.66
D	10	100	3	0.46%	0.49%	161.00	1741.80	6.92	6110.00	0.02%	3	2102.17
	10	100	4	0.54%	0.57%	162.20	1763.20	7.49	6669.80	0.00%	4	3324.51
	10	100	5	0.54%	0.54%	123.20	1473.00	6.20	3133.80	0.00%	5	1678.22
	20	100	3	1.62%	1.76%	58.80	1527.60	3.36	10217.60	0.12%	0	–
	20	100	4	1.52%	1.59%	50.40	1469.60	3.38	9584.20	0.07%	0	–
	20	100	5	1.58%	1.63%	41.20	1325.00	3.21	11828.80	0.04%	1	6608.36
	30	100	3	1.63%	1.80%	37.20	1700.40	2.78	7775.00	0.15%	0	–
	30	100	4	1.37%	1.51%	34.00	1680.40	2.95	9740.80	0.13%	0	–
	30	100	5	1.16%	1.22%	28.00	1576.60	2.94	10950.20	0.06%	0	–
Avg. (D)				1.16%	1.24%	77.33	1584.18	4.36	8445.58	0.07%	13	3428.31
E	10	100	3	0.41%	0.42%	181.80	1933.60	7.78	66.00	0.00%	5	63.52
	10	100	4	0.04%	0.04%	192.60	2026.40	9.06	1.20	0.00%	5	10.73
	10	100	5	0.28%	0.28%	189.00	1982.00	9.02	4.20	0.00%	5	14.40
	20	100	3	1.08%	1.10%	65.20	1617.40	3.51	123.60	0.00%	5	53.89
	20	100	4	0.52%	0.54%	65.00	1620.80	3.87	173.40	0.00%	5	80.72
	20	100	5	0.95%	0.97%	68.00	1636.60	4.39	136.80	0.00%	5	68.59
	30	100	3	1.84%	1.95%	32.40	1519.00	2.43	1698.00	0.00%	5	597.99
	30	100	4	1.03%	1.10%	33.00	1567.40	2.91	711.60	0.00%	5	274.41
	30	100	5	0.92%	0.98%	34.40	1585.60	3.38	478.80	0.00%	5	190.87
Avg. (E)				0.79%	0.82%	95.71	1720.98	5.15	377.07	0.00%	45	150.57
Avg. M = 100				0.68%	0.69%	82.28	1562.99	4.04	2940.93	0.02%	103	1193.85
C	15	200	4	0.00%	0.00%	357.80	2548.20	87.46	0.00	0.00%	5	87.46
	15	200	5	0.00%	0.00%	356.00	2131.40	54.79	0.00	0.00%	5	54.79
	30	200	4	0.00%	0.00%	47.00	2024.60	10.15	0.00	0.00%	5	10.15
	30	200	5	0.00%	0.00%	27.60	1579.00	10.73	0.00	0.00%	5	10.73
Avg. (C)				0.00%	0.00%	197.10	2070.80	40.78	0.00	0.00%	20	40.78
D	15	200	4	0.72%	0.75%	182.20	2160.00	68.73	5278.20	0.02%	0	–
	15	200	5	0.53%	0.54%	162.00	2767.00	58.26	4111.60	0.01%	3	1429.00
	30	200	4	1.19%	1.25%	63.60	2659.00	29.70	5289.00	0.06%	0	–
	30	200	5	0.94%	0.96%	60.00	2545.00	25.18	7500.80	0.02%	0	–
Avg. (D)				0.85%	0.88%	116.95	2532.75	45.47	5544.90	0.03%	3	1429.00
E (p = 0.8)	15	200	4	0.37%	0.37%	428.80	2779.80	123.18	16.20	0.00%	5	287.66
	15	200	5	0.38%	0.38%	412.40	1627.80	104.06	30.00	0.00%	5	433.27
	30	200	4	0.64%	0.65%	128.80	1711.80	45.11	148.20	0.00%	5	419.52
	30	200	5	0.41%	0.41%	137.40	1720.40	45.87	47.40	0.00%	5	206.73
Avg. (E, 0.8)				0.45%	0.45%	276.85	1959.95	79.55	60.45	0.00%	20	336.79
E (p = 1.0)	15	200	4	0.42%	0.42%	390.00	2219.00	106.92	53.40	0.00%	5	975.23
	15	200	5	0.23%	0.23%	405.00	1710.20	148.19	58.80	0.00%	5	1037.15
	30	200	4	0.41%	0.41%	128.00	1633.20	44.52	46.80	0.00%	5	199.19
	30	200	5	0.28%	0.28%	130.40	1689.80	44.47	86.40	0.00%	5	271.30
Avg. (E, 1.0)				0.33%	0.34%	263.35	1813.05	86.03	61.35	0.00%	20	620.71
Avg. M = 200				0.41%	0.42%	213.56	2094.14	62.96	1416.68	0.01%	63	606.82

Table 5.2: Experimental results for the branch-and-price algorithm

Instances					root node			whole search tree		
ID	Corr.	N	M	K	primal gap	p.d. gap	time(s)	final gap	ev. nodes	time(s)
c10400	C	10	200	2	43.51%	43.51%	643.21	0.00%	6	757.69
c20400		20	200	2	0.64%	0.69%	111.75	0.00%	30	278.82
c40400		40	200	2	0.54%	0.54%	20.35	0.00%	12	50.25
c15900		15	300	3	0.09%	0.09%	4514.97	0.00%	6	4854.12
c30900		30	300	3	0.10%	0.10%	372.94	0.00%	90	2078.95
c60900		60	300	3	0.00%	0.00%	26.40	0.00%	0	31.95
c201600		20	400	4	0.07%	0.07%	5381.14	0.00%	3	5877.77
c401600		40	400	4	0.00%	0.00%	203.58	0.00%	0	213.42
c801600		80	400	4	0.00%	0.00%	43.99	0.00%	0	61.22
Avg.					4.99%	5.00%	1257.59	0.00%	16.33	1578.24
d10400	D	10	200	2	0.46%	0.49%	526.17	0.03%	-	-
d20400		20	200	2	0.51%	0.58%	86.22	0.07%	-	-
d40400		40	200	2	1.33%	1.48%	23.48	0.15%	-	-
d15900		15	300	3	0.63%	0.65%	2576.19	0.02%	-	-
d30900		30	300	3	1.10%	1.19%	301.49	0.09%	-	-
d60900		60	300	3	0.93%	1.05%	101.30	0.12%	-	-
d201600		20	400	4	0.00%	0.76%	10606.39	0.76%	-	-
d401600		40	400	4	0.00%	0.74%	649.28	0.74%	-	-
d801600		80	400	4	1.00%	1.07%	218.10	0.07%	-	-
Avg.					0.66%	0.89%	1676.51	0.23%	-	-
e10400	E	10	200	2	1.41%	1.42%	670.55	0.00%	804	17473.80
e20400		20	200	2	0.21%	0.21%	94.91	0.00%	3	121.89
e40400		40	200	2	1.96%	1.97%	27.38	0.00%	39	151.86
e15900		15	300	3	0.03%	0.03%	4732.69	0.00%	15	6607.22
e30900		30	300	3	0.85%	0.85%	350.93	0.00%	72	1759.14
e60900		60	300	3	0.02%	0.02%	130.60	0.00%	12	313.62
e201600		20	400	4	0.15%	0.15%	12446.65	0.00%	3	12996.30
e401600		40	400	4	0.02%	0.02%	875.21	0.00%	45	3188.50
e801600		80	400	4	0.17%	0.17%	276.39	0.00%	213	2685.78
Avg.					0.54%	0.54%	2178.37	0.00%	134.00	5033.12

Table 5.3: Testing branch-and-price on large size instances

Instances				CPLEX 6.5.3			CPLEX + Logic Cuts			CPLEX + LC + Heurs			Branch and Price		
Corr.	N	M	K	gap	opt	time(s)	gap	opt	time(s)	gap	opt	time(s)	gap	opt	time(s)
C	10	100	3	0.00%	5	3.61	0.00%	5	6.56	0.00%	5	4.10	0.00%	5	5.19
	10	100	4	0.00%	5	8.20	0.00%	5	11.00	0.00%	5	8.16	0.00%	5	5.14
	10	100	5	0.00%	5	5.44	0.00%	5	7.50	0.00%	5	6.60	0.00%	5	4.77
	20	100	3	0.00%	5	4.49	0.00%	5	7.92	0.00%	5	6.60	0.00%	5	1.62
	20	100	4	0.00%	5	6.25	0.00%	5	7.96	0.00%	5	7.65	0.00%	5	1.58
	20	100	5	0.00%	5	12.64	0.00%	5	15.03	0.00%	5	14.06	0.00%	5	1.61
	30	100	3	0.00%	5	5.89	0.00%	5	6.99	0.00%	5	6.47	0.00%	5	1.18
	30	100	4	0.00%	5	8.08	0.00%	5	13.55	0.00%	5	12.24	0.00%	5	1.60
	30	100	5	0.00%	5	9.88	0.00%	5	11.43	0.00%	5	13.19	0.00%	5	1.28
Avg. (C)				0.00%	45	7.16	0.00%	45	9.77	0.00%	45	8.78	0.00%	45	2.66
D	10	100	3	0.64%	0	-	0.58%	0	-	0.24%	0	-	0.02%	3	2102.17
	10	100	4	0.57%	0	-	0.50%	0	-	0.28%	0	-	0.00%	4	3324.51
	10	100	5	0.66%	0	-	0.52%	0	-	0.22%	0	-	0.00%	5	1678.22
	20	100	3	1.48%	0	-	1.32%	0	-	0.90%	0	-	0.12%	0	-
	20	100	4	1.33%	0	-	1.30%	0	-	0.89%	0	-	0.07%	0	-
	20	100	5	1.28%	0	-	0.99%	0	-	0.89%	0	-	0.04%	1	6608.36
	30	100	3	1.87%	0	-	1.68%	0	-	1.32%	0	-	0.15%	0	-
	30	100	4	1.86%	0	-	1.68%	0	-	1.39%	0	-	0.00%	0	-
	30	100	5	1.51%	0	-	1.46%	0	-	1.18%	0	-	0.06%	0	-
Avg. (D)				1.24%	0	-	1.12%	0	-	0.81%	0	-	0.07%	13	3428.31
E	10	100	3	0.14%	2	791.06	0.14%	4	3820.26	0.03%	4	708.37	0.00%	5	63.52
	10	100	4	0.06%	3	521.65	0.08%	3	3191.33	0.02%	4	714.58	0.00%	5	10.73
	10	100	5	0.05%	4	466.01	0.04%	4	2278.66	0.00%	5	1734.20	0.00%	5	14.40
	20	100	3	1.77%	0	-	1.49%	0	-	0.86%	0	-	0.00%	5	53.89
	20	100	4	1.52%	0	-	1.71%	0	-	0.97%	0	-	0.00%	5	80.72
	20	100	5	1.58%	0	-	2.60%	0	-	2.04%	0	-	0.00%	5	68.59
	30	100	3	3.90%	0	-	4.02%	0	-	3.32%	0	-	0.00%	5	597.99
	30	100	4	2.97%	0	-	3.57%	0	-	3.33%	0	-	0.00%	5	274.41
	30	100	5	3.18%	0	-	3.04%	0	-	2.55%	0	-	0.00%	5	190.87
Avg. (E)				1.69%	9	592.91	1.86%	11	3096.75	1.46%	13	1052.38	0.00%	45	150.57
Avg. M = 100				0.98%	54	300.04	0.99%	56	1553.26	0.76%	58	530.58	0.02%	103	1193.85
C	15	200	4	0.00%	5	91.86	0.00%	5	93.90	0.00%	5	123.76	0.00%	5	87.46
	15	200	5	0.00%	5	82.64	0.00%	5	131.47	0.00%	5	103.48	0.00%	5	54.79
	30	200	4	0.00%	5	273.34	0.00%	5	385.82	0.00%	5	273.05	0.00%	5	10.15
	30	200	5	0.39%	4	243.10	0.00%	5	586.54	0.00%	5	143.33	0.00%	5	10.73
Avg. (C)				0.10%	19	172.73	0.00%	20	299.44	0.00%	20	160.91	0.00%	20	40.78
D	15	200	4	0.63%	0	-	0.54%	0	-	0.29%	0	-	0.02%	0	-
	15	200	5	0.60%	0	-	0.48%	0	-	0.22%	0	-	0.01%	3	1429.00
	30	200	4	0.91%	0	-	0.99%	0	-	0.79%	0	-	0.06%	0	-
	30	200	5	0.80%	0	-	0.83%	0	-	0.64%	0	-	0.02%	0	-
Avg. (D)				0.73%	0	-	0.71%	0	-	0.49%	0	-	0.03%	3	1429.00
E	15	200	4	0.17%	0	-	0.37%	0	-	0.17%	0	-	0.00%	5	287.66
p 0.8	15	200	5	0.17%	0	-	0.64%	0	-	0.16%	0	-	0.00%	5	433.27
	30	200	4	1.17%	0	-	2.15%	0	-	1.86%	0	-	0.00%	5	419.52
	30	200	5	1.10%	0	-	2.26%	0	-	2.03%	0	-	0.00%	5	206.73
Avg. (E, p 0.8)				0.65%	0	-	1.36%	0	-	1.05%	0	-	0.00%	20	336.79
E	15	200	4	0.25%	0	-	0.85%	0	-	0.66%	0	-	0.00%	5	975.23
p 1.0	15	200	5	0.24%	0	-	0.89%	0	-	0.80%	0	-	0.00%	5	1037.15
	30	200	4	1.06%	0	-	2.11%	0	-	2.28%	0	-	0.00%	5	199.19
	30	200	5	1.17%	0	-	1.97%	0	-	2.30%	0	-	0.00%	5	271.30
Avg. (E, p 1.0)				0.68%	0	-	1.45%	0	-	1.51%	0	-	0.00%	20	620.71
Avg. M = 200				0.54%	19	172.73	0.88%	20	299.44	0.76%	20	160.91	0.01%	63	606.82

Table 5.4: Comparison between CPLEX 6.5.3 and branch-and-price

Rounding heuristic:

```
Input:  $z_i^d \ \forall i \in \mathcal{N} \ \forall d \in \mathcal{D}'_i$
(the solution of R-LMP)
Output:  $i(j) \in \mathcal{N}$ (agent assignment) and $k(j) \in \mathcal{K}$ (level assignment) $\forall j \in \mathcal{M}$
(a feasible solution for the MGAP)

(Initialization)
forall $i \in \mathcal{N}$ do
        $C^i$ := $\emptyset$;  $q_i$ := $b_i$

(Step 1:  task-agent assignment)
$f_{ij} = \sum_{d \in \mathcal{D}'_i} (\sum_{k \in \mathcal{K}} x_{ijk}^d) z_i^d \quad \forall i \in \mathcal{N} \ \forall j \in \mathcal{M}$
forall $j \in \mathcal{M}$ do
        $I_j$ := $\{i \mid q_i \geq$ min $_{k \in \mathcal{K}}\{a_{ijk}\}\}$
        if $I_j$ = $\emptyset$ then FAIL
        else
              $i(j)$ := argmax$_{i \in I_j}\{f_{ij}\}$
              $C^{i(j)}$ := $C^{i(j)} \cup \{j\}$;  $q_{i(j)}$ := $q_{i(j)} -$ min $_{k \in \mathcal{K}}\{a_{i(j)jk}\}$

(Step 2:  task-level assignment)
forall $i \in \mathcal{N}$ do
        Solve to optimality the MCKP:

                min $\sum_{j \in C^i} \sum_{k \in K} c_{ijk} x_{ijk}$
                s.t. $\sum_{j \in C^i} \sum_{k \in \mathcal{K}} a_{ijk} x_{ijk} \leq b_i$
                     $\sum_{k \in \mathcal{K}} x_{ijk} = 1$            $\forall j \in C^i$
                     $x_{ijk} \in \{0,1\}$           $\forall j \in C^i, \quad \forall k \in \mathcal{K}$

        forall $j \in C^i$ do $k(j)$ := $(k \mid x_{ijk} = 1)$
```

Local search:

```
(Shift Neighborhood)
forall $j \in \mathcal{M}$ do
       $i'$ := (null element); $v$ := $c_{i(j)jk(j)}$
       forall $i \in \mathcal{N}$, forall $k \in \mathcal{K}$ do
              if $(c_{ijk} < v$ AND $q_i \geq a_{ijk})$ then
                 $i'$ := $i$; $k'$ := $k$; $v$ := $c_{ijk}$

       if $(i' \neq$ (null element)$)$ then
          $q_{i(j)}$ := $q_{i(j)} + a_{i(j)jk(j)}$
          $q_{i'}$ := $q_{i'} - a_{i'jk'}$
          $i(j)$ := $i'$; $k(j)$ := $k'$

(Swap Neighborhood)
forever do

        $i'$ := (null element); $v$ := $0$
        forall $js, jd \in \mathcal{N}$ do
               forall $ks, kd \in \mathcal{K}$ do
                      $is$ := $i(js)$; $id$ := $i(jd)$
                      $\Delta$ := $(c_{id\ js\ ks} + c_{is\ jd\ kd}) - (c_{is\ js\ k(js)} + c_{id\ jd\ k(jd)})$
                      $\delta_s$ := $a_{is\ jd\ kd} - a_{is\ js\ k(js)}$
                      $\delta_d$ := $a_{id\ js\ ks} - a_{id\ jd\ k(jd)}$
                      if $(\Delta < v$ AND $\delta_s \leq q_{is}$ AND $\delta_d \leq q_{id})$ then
                         $v$ := $\Delta$;
                         $j'$ := $js$; $j''$ := $jd$
                         $k'$ := $ks$; $k''$ := $kd$

        if $(i' =$ (null element)$)$ then BREAK

        $q_{i(j')}$ := $q_{i(j')} - a_{i(j')j'k(j')} + a_{i(j')j''k''}$
        $q_{i(j'')}$ := $q_{i(j'')} - a_{i(j'')j''k(j'')} + a_{i(j'')j'k'}$

        SWAP$(i(j'), i(j''))$; $k(j')$ := $k'$; $k(j'')$ := $k''$
```

Part III

Location Problems

Finally, we consider single-source location problems, in which a set of customers has to be partitioned in regions, and a facility must be activated in each region. In our view, these problems combine the characteristics of packing and assignment problems. In fact, both fixed costs for activating the facilities and allocation costs for assigning customers to facilities interact in the optimization process.

In Chapter 6 we describe a branch-and-price algorithm for the capacitated p-median problem, which was the starting point of our study. The main focus is on the experimental properties of the algorithm: we compare different multiple pricing techniques and give quantitative evaluations of each component of the algorithm.

Then, in Chapter 7, we extend this approach to the broad class of single-source capacitated location problems. We review the existing literature and give modeling motivations to our work. The aim is to assess the effectiveness of a prototype of a general purpose solver for this class of problems, based on branch-and-price. In fact, we conduct an extensive experimental campaign with two objectives in mind. First, highlighting the strong and weak features of this kind of approach, when compared to the use of a branch-and-cut based package. Second, discussing the peculiarities of location models from an experimental point of view.

Chapter 6

A branch-and-price algorithm for the capacitated p-median problem

The capacitated p-median problem is the variation of the well-known p-median problem in which a demand is associated to each user, a capacity is associated to each candidate median and the total demand of the users associated to the same median must not exceed its capacity. We present a branch-and-price algorithm, that exploits column generation, heuristics and branch-and-bound to compute optimal solutions. We compare our branch-and-price algorithm with other methods proposed so far and we present computational results both on test instances taken from the literature and on random instances with different values of the ratio between the number of medians and the number of users.

6.1 Introduction

The p-median problem (PMP) is one of the most widely studied problems in location theory. It consists of partitioning the vertices of a given graph into p subsets and to choose a *median* vertex in each subset, minimizing the sum of the distances between each vertex of the graph and the median of its subset. The PMP arises in many different contexts such as network design, telecommunications, distributed database design, transportation and distribution logistics. Kariv and Hakimi [53] proved that the PMP is $\mathcal{NP}$-hard. Optimization algorithms based on Lagrangean relaxation have been proposed in [80], [24], [21] and [9]; approaches based on dual formulations are illustrated in [40] and [46]. A survey on the PMP can be found in [61].

If medians represent facilities providing a certain service and each vertex of the graph represents a user who requires that service, it is natural to introduce capacity constraints, so that the sum of the demands of the users assigned to each facility is forced not to exceed the capacity of that facility. This yields the capacitated p-median problem (CPMP). Algorithms devised for the uncapacitated PMP cannot be adapted to the CPMP in a straightforward way: even finding a feasible solution is $\mathcal{NP}$-complete when capacities are considered.

Very recently, two heuristic algorithms for the CPMP have been presented: Lorena and Senne [69] followed a column generation approach, finding good solutions on real instances with up to 402 vertices, while Diaz and Fernández [29] attacked an instance with 737 vertices through hybrid scatter search and path relinking.

In this paper we deal with exact optimization algorithms for the CPMP. The optimization of the CPMP can be pursued through the adaptation of an algorithm developed by Pirkul [86] for the capacitated concentrator location problem (CCLP): the CCLP is similar to the CPMP in that it involves the allocation of a set of indivisible users' demands to capacitated facilities, but the number of facilities to be used is not fixed; instead a cost is incurred for each facility used. The algorithm is based on Lagrangean relaxation and branch-and-bound and it can solve the CCLP to optimality on graphs with up to 100 vertices in a few minutes. Another approach was suggested by Ross and Soland [91] and consists of the reformulation of CPMP instances as generalized assignment problem (GAP) instances; though based on an elegant transformation, this technique is not competitive because the GAP instances it produces are very large. More recently, Baldacci et al. [6] developed an algorithm which either proves optimality or provides a bound on the approximation obtained; the authors report about experiments for problems with up to 200 vertices and 20 medians, when their algorithm is initialized with quasi-optimal feasible solutions. In this paper we present a branch-and-price algorithm for the exact solution of the CPMP and we compare it with a general purpose integer linear programming solver (CPLEX) and with the algorithm of Pirkul (adapted to the CPMP). We also compare our results with those obtained by the algorithm of Baldacci et al. [6] when it is properly initialized.

The paper is organized as follows. In Section 6.2 we define the mathematical model of the CPMP and we present a disaggregated formulation; then we compare the lower bounds obtained from the corresponding linear relaxations; in Section 6.3 we describe our branch-and-price algorithm and we illustrate some implementation details; in Section 6.4 we describe the algorithms we used as benchmarks and we present experimental results.

6.2 Formulations

Consider a graph $G = (\mathcal{N}, E)$ and a subset $\mathcal{M} \subseteq \mathcal{N}, \mathcal{M} = \{1 \dots M\}$ of its vertices that are candidates to be medians. For each vertex $i \in \mathcal{N}$ an integer weight w_i represents its demand. For each candidate median $j \in \mathcal{M}$ an integer coefficient Q_j represents its capacity. Integer coefficients d_{ij} (usually referred to as *distances*) describe the cost of allocating each vertex $i \in \mathcal{N}$ to each median $j \in \mathcal{M}$. We make the assumption that $d_{ij} \geq 0 \ \forall i \in \mathcal{N}, j \in \mathcal{M}$. The vertex set $\mathcal{N}$ must be partitioned into p subsets (*clusters*), where p is given. The capacitated p-median problem (CPMP) is formulated as follows:

$$\begin{aligned}
CPMP) \quad \min \quad & v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} & & \\
\text{s.t.} \quad & \sum_{j \in \mathcal{M}} x_{ij} = 1 & \forall i \in \mathcal{N} & \qquad (6.1) \\
& \sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j & \forall j \in \mathcal{M} & \qquad (6.2) \\
& \sum_{j \in \mathcal{M}} y_j = p & & \qquad (6.3) \\
& x_{ij} \in \{0, 1\} & \forall i \in \mathcal{N}, \ \forall j \in \mathcal{M} & \\
& y_j \in \{0, 1\} & \forall j \in \mathcal{M} &
\end{aligned}$$

Binary variables x are assignment variables: $x_{ij} = 1$ if and only if vertex i is assigned to a median located in vertex j. Binary variables y correspond to location decisions: $y_j = 1$ if and only if vertex j is selected to be a median. The objective is to minimize the sum of allocation costs. Set partitioning constraints (6.1) impose that each vertex is assigned to a median. Capacity constraints (6.2) impose that the sum of the vertex weights in each cluster do not exceed the capacity of the median and forbid the assignment of vertices to unselected medians. To satisfy constraint (6.3) exactly p medians must be selected.

The linear relaxation of this formulation can be strengthened in two ways: first, including the inequalities $x_{ij} \leq y_j$, $\forall i \in \mathcal{N}, j \in \mathcal{M}$, arising from constraints (6.2); second, by dynamic generation of valid inequalities. The illustration of the test instances we have used is presented in Subsection 6.2.4. Our computational experience using CPLEX 6.5 as a general purpose solver is reported in Section 6.4.

6.2.1 Dantzig-Wolfe decomposition

Hereafter we derive an alternative formulation of the CPMP amenable to a column generation approach.

Let $(\mathbf{1}, p)$ be the vector of the right-hand side terms of constraints (6.1) and (6.3), $\boldsymbol{d^j} = (d_{1j} \dots d_{Nj})$ be the vector of the distances between each vertex and the candidate median $j \in \mathcal{M}$, $\boldsymbol{x^j} = (x_{1j}, \dots, x_{Nj})$ the vector of assignment variables related to the median in $j \in \mathcal{M}$ and $\boldsymbol{w} = (w_1, \dots, w_N)$ the vector of vertex weights. Consider the CPMP linear relaxation, expressed as follows:

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{M}} (\boldsymbol{d^j}, 0)^T (\boldsymbol{x^j}, y_j) & \\ \text{s.t.} \quad & \sum_{j \in \mathcal{M}} (\boldsymbol{x^j}, y_j) = (\mathbf{1}, p) & \\ & (\boldsymbol{x^j}, y_j) \in \Omega^j & \forall j \in \mathcal{M} \end{aligned}$$

where $\Omega^j = \{(\boldsymbol{x^j}, y_j) \in \Re_+^{N+1} \mid x_{ij} \leq 1 \ \forall i \in \mathcal{N}, \ y_j \leq 1, \ \boldsymbol{w}^T \boldsymbol{x^j} \leq Q_j y_j\}$. The optimal value of this linear relaxation is a lower bound for the CPMP. In order to improve this bound, we replace each polyhedron Ω^j with the convex hull of its integer points:

$$\begin{aligned} \min \quad & \sum_{j \in \mathcal{M}} (\boldsymbol{d^j}, 0)^T (\boldsymbol{x^j}, y_j) & \\ \text{s.t.} \quad & \sum_{j \in \mathcal{M}} (\boldsymbol{x^j}, y_j) = (\mathbf{1}, p) & \\ & (\boldsymbol{x^j}, y_j) \in \text{conv}(\Omega^j) & \forall j \in \mathcal{M} \end{aligned}$$

Since Ω^j is the polyhedron of the linear relaxation of a binary knapsack problem (see [74] for a classical reference), which is known not to have the integrality property, its extreme points can have fractional coordinates; therefore the lower bound computed after the convexification of each Ω^j can be stronger than that of the linear relaxation of the CPMP (and our experiments reported in table 6.1 show that this is actually the case).

Since the y variables are bounded and since the weights and the capacities are non-negative, every polyhedron Ω^j is bounded, and every solution $(\boldsymbol{x^j}, y_j) \in \text{conv}(\Omega^j)$ can be obtained as a convex combination of the $L_j + 1$ extreme points of $\text{conv}(\Omega^j)$. We indicate the set of the extreme points with $\{(\mathbf{0}, 0)), (\bar{\boldsymbol{x}}^{\mathbf{1}}, 1), \dots, (\bar{\boldsymbol{x}}^{\boldsymbol{L_j}}, 1)\}$. Therefore we have

$$(\boldsymbol{x^j}, y_j) = \sum_{k \in Z^j} (\bar{\boldsymbol{x}}^{\boldsymbol{k}}, \bar{y}^k) z_k^j, \quad \sum_{k \in Z^j} z_k^j = 1, \ \ z_k^j \in \Re_+ \ \ \forall k \in Z^j \tag{6.4}$$

where each $k \in Z^j$ is the index of an extreme point of Ω^j. Substituting expression

(6.4) in the formulation of the linear relaxation of the CPMP we obtain

$$\begin{aligned}
\min \quad & \sum_{j\in\mathcal{M}} (\boldsymbol{d^j}, 0)^T \sum_{k\in Z^j} (\boldsymbol{\bar{x}^k}, \bar{y}^k) z_k^j \\
\text{s.t.} \quad & \sum_{j\in\mathcal{M}} \sum_{k\in Z^j} (\boldsymbol{\bar{x}^k}, \bar{y}^k) z_k^j = (\mathbf{1}, p) \\
& \sum_{k\in Z^j} z_k^j = 1 && \forall j \in \mathcal{M} \\
& z_k^j \in \Re_+ && \forall j \in \mathcal{M} \ \ \forall k \in Z^j
\end{aligned}$$

Therefore the decomposition and convexification of the linear relaxation of the CPMP yields the following linear master problem (LMP):

$$\begin{aligned}
LMP) \quad \min \quad & \sum_{j\in\mathcal{M}} \sum_{k\in Z^j} \Big(\sum_{i\in\mathcal{N}} d_{ij} x_i^k\Big) z_k^j \\
\text{s.t.} \quad & \sum_{j\in\mathcal{M}} \sum_{k\in Z^j} x_i^k z_k^j \geq 1 && \forall i \in \mathcal{N} && (6.5) \\
& -\sum_{j\in\mathcal{M}} \sum_{k\in Z^j} z_k^j \geq -p && && (6.6) \\
& -\sum_{k\in Z^j} z_k^j \geq -1 && \forall j \in \mathcal{M} && (6.7) \\
& z_k^j \in \Re_+ && \forall j \in \mathcal{M}, \ \ \forall k \in Z^j.
\end{aligned}$$

Each column of this model corresponds to a feasible cluster, that is an assignment of vertices to a median, that satisfies the capacity constraint. Each cluster k is described by assignment coefficients x_i^k equal to 1 if and only if vertex $i \in \mathcal{N}$ belongs to cluster $k \in Z^j$ of median $j \in \mathcal{M}$. A binary variable z_k^j is associated with each cluster. Constraints (6.5) guarantee that each vertex is assigned to at least one median; constraint (6.6) implies that the total number of selected clusters is at most p; constraints (6.7) impose that no more than one cluster is associated to the same median.

In model (6.5)-(6.7) all equality constraints have been replaced by inequalities and hereafter we briefly discuss the correctness and usefulness of the substitutions.

Partitioning constraints (6.1) can be replaced by covering constraints (6.5) because all distances are non-negative and therefore it does always exist an optimal solution in which no user is assigned more than once.

Since there is no fixed cost associated to the medians, if an optimal solution exists with $p' < p$ medians, there is also an equivalent solution with p medians, $p-p'$ of which have empty clusters. Therefore equality constraint (6.3) can be replaced by the inequality $\sum_{j\in\mathcal{M}} y_j \leq p$. This yields the constraint $\sum_{j\in\mathcal{M}} \sum_{k\in Z^j} y^k z_k^j \leq p$ in our reformulation. However empty clusters do affect neither the objective function value nor the satisfaction of constraints (6.5) and therefore we can make the assumption that no empty cluster (i.e. a cluster with $y^k = 0$) appears in LMP. For this reason the constraint $\sum_{j\in\mathcal{M}} \sum_{k\in Z^j} y^k z_k^j \leq p$ can be restated as $\sum_{j\in\mathcal{M}} \sum_{k\in Z^j} z_k^j \leq p$ and constraints (6.7) are stated as inequalities.

Each set Z^j of feasible clusters contains an exponential number of elements. Since LMP cannot be solved directly because of the exponential number of its columns, column generation (see [43]) is applied: a restricted linear master problem (RLMP), defined by a relatively small subset of columns is considered and solved to optimality; then, a search is performed for new columns of negative reduced cost and if any such column is found, it is inserted into the formulation and the RLMP is solved again. When no columns of negative reduced cost exist, the optimal solution of the RLMP is also optimal for the LMP and its value is a valid dual bound to be used in a branch-and-bound framework.

When solving LMP by column generation, the set covering formulation above has at least two advantages compared to the equivalent set partitioning formulation: first, the generation of feasible solutions is easier and therefore the primal simplex algorithm can be always easily initialized with a feasible basis; second, all dual variables are non-negative and this restricts the dual space and speeds up the convergence. A drawback is that the LMP may have an optimal fractional solution, in which some user is covered more than once; in this case some minor modifications to the primal bounding procedure and to the branching scheme have to be made (see Section 6.3).

6.2.2 The pricing problem

Let $\boldsymbol{\lambda} \in R_+^N$, $\eta \in R_+$ and $\boldsymbol{\mu} \in R_+^M$ be the vectors of non-negative dual variables corresponding to constraints (6.5), (6.6) and (6.7) respectively; the reduced cost of column $k \in Z^j$ is

$$r^k(\boldsymbol{\lambda}, \eta, \boldsymbol{\mu}) = \sum_{i\in\mathcal{N}} d_{ij} x_i^k - \sum_{i\in\mathcal{N}} \lambda_i x_i^k + \eta + \mu_j$$

To find columns with negative reduced cost, one must solve a pricing problem for each median $j \in \mathcal{M}$:

$$\begin{aligned} \min \quad & \sum_{i\in\mathcal{N}} (d_{ij} - \lambda_i) x_i^k + \eta + \mu_j \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}} w_i x_i^k \leq Q_j \\ & x_i^k \in \{0,1\} \qquad\qquad \forall i \in \mathcal{N} \end{aligned}$$

and this requires the solution of the following binary knapsack problem:

$$
\begin{aligned}
KP_j)\ \ \max\ \ & \tau_j = \sum_{i \in \mathcal{N}} (\lambda_i - d_{ij}) x_i^k \\
\text{s.t.}\ \ & \sum_{i \in \mathcal{N}} w_i x_i^k \leq Q_j \\
& x_i^k \in \{0,1\} && \forall i \in \mathcal{N}
\end{aligned}
$$

Therefore the computational effectiveness of our branch-and-price algorithm mainly relies upon that of the algorithms used for the iterated solution of the master problem (i.e. the simplex algorithm) and the binary knapsack subproblem. One of our main motivations for applying the branch-and-price technique to the CPMP is the existence of very effective algorithms and reliable implementations for both linear programming and the binary knapsack problem.

6.2.3 Lagrangean relaxation

The lower bound obtained from the LMP can also be obtained through the Lagrangean relaxation of semi-assignment constraints (6.1) of the CPMP formulation:

$$
\begin{aligned}
LR)\ \min\ \ & \omega_{LR} = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} + \sum_{i \in \mathcal{N}} \lambda_i (1 - \sum_{j \in \mathcal{M}} x_{ij}) \\
\text{s.t.}\ \ & \sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j && \forall j \in \mathcal{M} && (6.8) \\
& \sum_{j \in \mathcal{M}} y_j = p && && (6.9) \\
& x_{ij} \in \{0,1\} && \forall i \in \mathcal{N}, \forall j \in \mathcal{M} \\
& y_j \in \{0,1\} && \forall j \in \mathcal{M}
\end{aligned}
$$

The Lagrangean multipliers in LR correspond to the dual variables $\boldsymbol{\lambda}$ in the LMP and the Lagrangean subproblem can be decomposed in the same M binary knapsack problems as the pricing subproblem in the column generation approach (for the equivalence between Dantzig-Wolfe decomposition and Lagrangean relaxation the reader is referred to mathematical programming textbooks like [83]). Therefore column generation can be used as a method alternative to subgradient optimization to update the Lagrangean multipliers.

6.2.4 Benchmark instances and lower bound comparison

We considered a testbed made of four classes of instances, named α, β, γ and δ. Each class consists of forty instances: twenty of them are taken from the literature

([84], [70] and [6]) and concern graphs with 50 and 100 vertices and $p = \frac{N}{10}$; the other twenty instances were randomly generated on graphs with cardinality 150 (15 medians) and 200 (20 medians): vertex coordinates were drawn from a uniform distribution between 1 and 100 and vertex demands were drawn from a uniform distribution between 1 and 20. Each d_{ij} coefficient was taken as the Euclidean distance computed from the coordinates of vertices i and j, rounded down to the nearest integer. In random instances all capacities were fixed to 120. In both the random and the literature instances, the set $\mathcal{N}$ of vertices and the set $\mathcal{M}$ of candidate medians coincide, hence $N = M$. In our experiments we could observe that the behavior of the different algorithms may be significantly influenced by the ratio between p and N. This means that a sound test set for the CPMP cannot be limited to instances where the ratio between p and N is constant. Therefore the same forty instances were solved with different number of medians: $p = \frac{N}{10}$ in class α, $p = \lfloor \frac{N}{4} \rfloor$ in class β, $p = \lfloor \frac{N}{3} \rfloor$ in class γ and $p = \lfloor \frac{2N}{5} \rfloor$ in class δ. The overall capacity was preserved in all classes by setting $Q_j = \lceil 12\frac{N}{p} \rceil \ \forall j \in \mathcal{M}$. In all instances in our test set the medians have the same capacity and $d_{ij} = 0$ whenever user i and median j coincide.

All tests presented in this paper were done on an Intel Pentium IV 1600 MHz PC with 512 MB RAM in Linux environment (RedHat 7.2, kernel 2.4.19). The algorithms were coded in C++, compiled with gcc/g++ version 2.96 with "-O3" optimization level. We imposed some resource limitation to every test: computation was halted after one hour or in case of memory overflow. ILOG CPLEX 6.5 was used as an LP solver in the branch-and-price algorithm.

In table 6.1 we compare (a) the bound obtained from the CPMP linear relaxation strengthened by inequalities $x_{ij} \leq y_j$ and tightened with clique, cover and GUB-cover inequalities automatically generated by CPLEX; (b) the bound obtained from Lagrangean relaxation and subgradient optimization (see Section 4.1 for details); (c) the bound obtained from the LMP. As a measure of the quality of these relaxations we indicate the average gap between each lower bound (LB) and the best known upper bound (UB), that it $\frac{UB-LB}{UB}$. The upper bounds we used are often the optimal solution values.

Bound (a) is systematically worse than the others on all classes and all sizes. Lower bound (b) is found by subgradient optimization halted after 300 iterations. Therefore it does not exactly correspond to lower bound (c), obtained from the LMP, that corresponds to solving the Lagrangean dual to optimality. Lower bound (c) is definitely dominating both (a) and (b); this can be interpreted as a measure of the effectiveness of the convexification of the capacity constraints.

N	p	CPMP (a)	SG (b)	LMP (c)
50	5	1.21%	1.14%	0.99%
	12	4.17%	3.20%	2.88%
	16	3.90%	2.81%	2.39%
	20	4.44%	4.04%	1.73%
100	10	1.26%	1.06%	0.90%
	25	3.84%	3.28%	2.90%
	33	3.43%	2.98%	2.67%
	40	3.84%	3.16%	2.12%
150	15	0.52%	0.58%	0.30%
	37	2.31%	1.93%	1.65%
	50	3.00%	2.54%	2.26%
	60	3.36%	2.85%	2.03%
200	20	0.64%	0.64%	0.44%
	50	4.89%	4.34%	3.93%
	66	5.38%	4.69%	4.32%
	80	5.48%	4.65%	3.22%

Table 6.1: Comparison between different lower bounds

6.3 A branch-and-price algorithm

6.3.1 Branching

The choice of the branching rule in branch-and-price algorithms is crucial because the addition of constraints may change the structure of the pricing problem. We have tested two branching strategies, outlined hereafter.

Strategy 1: branching on binary variables. This branching strategy is similar to the one used by Pirkul in his Lagrangean-based branch-and-bound algorithm for the capacitated concentrator location problem [86] and to the method by Diaz and Fernández for the single-source capacitated plant location problem [28]. Intuitively, fixing location variables y has much a stronger effect than fixing assignment variables x: when a location variable is fixed to 0 many assignment variables can be also fixed to 0; moreover, once p location variables have been fixed to 1, the CPMP reduces to a generalized assignment problem. Therefore it is effective to fix y variables early in the exploration of the search tree.

A candidate median j^* is selected and two possibilities are considered: in one branch all columns k with $k \in Z^{j^*}$ are discarded (that is the candidate median is fixed as "not used"); in the other branch the equality constraint $\sum_{k \in Z^{j^*}} z_k^{j^*} = 1$ is inserted into the RLMP (that is the candidate median is fixed as "used"). In both cases the structure of the pricing problem is not affected. The branching variable j^* is selected among the candidate medians not yet fixed as the one with minimum value of τ_j (that is the most unpromising one) and the branch in which the median is not used is explored first, with a depth-first policy.

When p medians have been fixed as "used" or $N - p$ have been fixed as "not used", branching is performed by fixing variables x, that is assigning the vertices to the selected medians. Assigning vertex i^* to median j^* corresponds to fixing $x^k_{i^*} = 1$ in the pricing subproblem KP_{j^*} and $x^k_{i^*} = 0$ in all KP_j, $j \neq j^*$. The branching vertex i^* is the one with the largest number of fractional assignments to different medians in the optimal solution of the LMP (instead, in Pirkul's algorithm the vertex with maximum weight w_i is chosen).

In each successor node of the search tree the branching vertex i^* is assigned to a different median, so that p branches are considered (in random order).

We used this branching strategy in conjunction with depth-first search policy, because this allows to exploit the structure of the LMP (namely, the columns and the optimal basis) of each predecessor node in the search tree to solve the LMP of its first successor without explicitly storing such information for each node. Moreover depth-first search policy quickly produces good feasible solutions, that can be useful to prune the search tree.

Strategy 2: branching on semi-assignment constraints. This branching strategy is similar to that used by Savelsbergh in his branch-and-price algorithm for the GAP [95]. For each vertex $i \in \mathcal{N}$ we consider the set $\mathcal{M}_i$ of candidate medians $j \in \mathcal{M}$ for which there is a fractional assignment x_{ij} in the optimal solution of the RLMP. In Savelsbergh's algorithm the set $\mathcal{M}_i$ is partitioned into two subsets $\mathcal{M}^-_i = \{j \in \mathcal{M} : x_{ij} > 0, j \leq j^*\}$ and $\mathcal{M}^+_i = \{j \in \mathcal{M} : x_{ij} > 0, j > j^*\}$ by choosing j^* in such a way that $|\mathcal{M}^-_i| = \lceil \frac{|\mathcal{M}_i|}{2} \rceil$. We elaborated on this idea, aiming at a balanced partition: the elements in $\mathcal{M}_i$ are sorted by non-increasing value of fractional assignment and they are inserted alternately in $\mathcal{M}^-_i$ and in $\mathcal{M}^+_i$. The set of candidate medians to which vertex i is not assigned is also partitioned into two subsets $\widehat{\mathcal{M}}^-_i$ and $\widehat{\mathcal{M}}^+_i$ of balanced cardinality. In [95] the author proposed to choose j^* as close as possible to $\frac{M}{2}$ with the constraint of leaving at least one fractional variable on each side, but we found that our choice of j^* produces better results. The vertex i^* selected for branching is the one for which $|\mathcal{M}_i|$ is maximum. In case of ties we select the vertex for which $\sum_{j \in \mathcal{M}^-_{i^*}} \sum_{k \in Z^j} x^k_{i^*} z^j_k$ is closest to $\frac{1}{2} \sum_{j \in \mathcal{M}_{i^*}} \sum_{k \in Z^j} x^k_{i^*} z^j_k$. Then we branch on the original constraint $\sum_{j \in \mathcal{M}} x_{i^*j} = 1$ by setting $\sum_{j \in \mathcal{M}^-_{i^*} \cup \widehat{\mathcal{M}}^-_{i^*}} x_{i^*j} = 0$ in one branch and $\sum_{j \in \mathcal{M}^+_{i^*} \cup \widehat{\mathcal{M}}^+_{i^*}} x_{i^*j} = 0$ in the other. The addition of these constraints does not change the structure of the knapsack problems; it only reduces their size.

Our experiments showed that branching strategy 2 is definitely more effective than strategy 1: only 57 of the 160 instances considered were solved to proven optimality with strategy 1 while 92 instances were solved with strategy 2. Moreover, on the instances solved with both methods, the algorithm with strategy 2 was faster. For this reason the computational results reported in tables 6.3 to 6.6

are referred to strategy 2.

6.3.2 Primal bound

The problem of finding a feasible solution to the CPMP is $\mathcal{NP}$-complete: the reduction from Partition [54] is the same as for the GAP (see [74]). To compute approximate solutions to the CPMP we modified the MTHG algorithm [73], that was devised by Martello and Toth for the GAP. Once defined suitable coefficients f_{ij} as a measure of the desirability of assigning each job i to each machine j, the MTHG algorithm goes through two steps. In the first step all jobs are sorted in decreasing order of a regret value and then they are assigned one at a time to their "most desired" machine. The regret is defined as the difference between the coefficient $f_{ij'}$ referred to the most desirable machine, and the coefficient $f_{ij''}$ referred to the second most desirable one. In the second step, provided that all jobs have been assigned without exceeding capacities, the solution is improved by single-job exchanges in a local search fashion. If some job remains unassigned the algorithm fails.

Jörnsten and Näsberg [52] proposed a similar algorithm in which all jobs are assigned even if capacity constraints are violated. Afterwards local search steps are executed to possibly achieve feasibility, followed by further local search steps to improve the solution.

In our algorithm we first choose p candidate medians (step 1); then the vertices are assigned to the medians in the same way as in the MTHG algorithm, that is without exceeding capacities (step 2); if some vertex remains unassigned, local search iterations are performed to produce a feasible solution (step 3); if this step succeeds, a final local search tries to improve the solution (step 4). A synthetic description of each step follows; the complete pseudo-code is reported in the appendix.

Step 1: medians selection. If a CPMP instance with non-uniform capacities is feasible, obviously it has at least one feasible solution in which the p most capable vertices are the medians. However in the CPMP instances considered in the literature all candidate medians have the same capacity. In this case, following Savelsbergh [95], the desirability coefficients have been defined as

$$f_{ij} = \sum_{k \in Z^j} x_i^k z_k^j \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M}$$

in order to produce an integer solution similar to the optimal solution of the RLMP. Then the p vertices with the highest values of ψ_j are chosen as medians, where

$$\psi_j = \sum_{i \in \mathcal{N}} f_{ij} \quad \forall j \in \mathcal{M}.$$

Step 2: direct assignment. In step 2 our algorithm is identical to that of Martello and Toth: the vertices are assigned to the selected medians in decreasing order of the regret value. Step 2 terminates as soon as a vertex is encountered which cannot be assigned to any median.

Step 3: assignment through exchanges. In this step all best-fit 1-exchanges are evaluated: an unassigned vertex i replaces a vertex k of smaller weight in a cluster C_j whenever the capacity constraint allows for such an exchange. The replaced element k is immediately inserted in another cluster if possible, as in step 2; otherwise it remains unassigned. The algorithm chooses i, j and k so that the residual capacity of C_j is minimized.

Step 4: solution improvement. Two different neighborhoods are explored in this final local search step. At each iteration one element is shifted from a cluster to another or two elements belonging to different clusters are swapped.

At the root node the evaluation of the primal bound is done after each column generation iteration; in all the other nodes of the search tree it is done at the end of the column generation routine.

6.3.3 Columns management

At each iteration the algorithm inserts into the RLMP all columns with negative reduced cost which have been found. After the termination of the column generation algorithm, it removes from the RLMP all columns whose reduced cost is higher than a threshold, that is a function of the gap between the best incumbent primal solution and the optimal solution of the LMP. In particular we set the threshold equal to the ratio between the primal-dual gap and the number of medians p. The removal is also performed during the execution of the column generation algorithm every time the number of columns exceeds a limit (set to 3000 in our experiments). In this case, the dual bound of the father node is used in the computation of the threshold.

In general there is no guarantee that the columns removed from the RLMP in one node of the search tree will not belong to the optimal solution of another node. Therefore it is useful to store removed columns in a pool and to scan it before running the pricing algorithm. Since every column is related to a particular candidate median, the pool is partitioned into M subsets. If any column with negative reduced cost is found in the pool, it is inserted into the RLMP.

To keep the pool size limited, the columns are erased from the pool when their reduced cost is non-negative for a certain number of consecutive evaluations (3 in our experiments).

To achieve feasibility the RLMP is initialized at each node with a dummy column of cost $\sum_{i \in \mathcal{N}} \max_{j \in \mathcal{M}} \{d_{ij}\}$, corresponding to an infeasible cluster, that covers all vertices and uses none of the free medians (those whose corresponding y variable has not been fixed). The structure of the dummy column is the following: all its coefficients in constraints (6.5) are set to 1; the coefficient in constraint (6.6) is set to 0; the coefficients in constraints (6.7) are set to -1 if the corresponding y variable has been fixed to 1, and they are set to 0 otherwise.

To improve the performance of the column generation algorithm at the root node, we also generate twenty initial solutions with the same primal heuristic illustrated above. Ten solutions are computed using $f_{ij} = -d_{ij}$ and ten using $f_{ij} = 1/d_{ij}$ for $i \neq j$ and setting f_{ii} to a very large value. In both cases the set of medians in step 1 is chosen at random with uniform probability distribution. The columns generated in this way are inserted even if the corresponding solutions are infeasible because of violations of the partitioning constraints.

In each node of the search tree the RLMP is initialized with the columns of the most recently solved node plus some additional columns taken from the pool with the following procedure: for every node of the search tree we store the optimal values of the dual variables; for each successor node we use the dual values of its predecessor to compute the reduced costs of all columns in the pool; if any column is found with negative reduced cost, it is added to the initial formulation of the RLMP in the successor node.

6.3.4 Lower bound and termination

The equivalence between Dantzig-Wolfe decomposition and Lagrangean relaxation is exploited both for variable fixing purposes and in the termination test.

At each iteration t of column generation the current values of the dual variables $\boldsymbol{\lambda}^t$ are used as Lagrangean multipliers to compute a valid lower bound:

$$\omega_{LR}^t = -\sum_{j \in \mathcal{M}^{LR}} \tau_j^t + \sum_{i \in \mathcal{N}} \lambda_i^t$$

where $\mathcal{M}^{LR} \subseteq \mathcal{M}$ is the set of vertices with the p maximum values of τ_j^t. In this way a sequence of valid lower bounds is computed during column generation and this allows to fix variables or even to prune the current node of the search tree before column generation is over.

When the gap between the optimal value of the RLMP at iteration t and the best incumbent Lagrangean lower bound $\omega_{LR}^{*t} = \max \ \{\omega_{LR}^1, \dots, \omega_{LR}^t\}$ is smaller than a predefined threshold (set to 10^{-4} in our tests), the column generation algorithm is terminated and ω_{LR}^{*t} is kept as the final lower bound. This allows to reduce the typical and undesired tailing-off effect in the column generation routine. This technique is not used at the root node, where column generation is iterated until no more columns with negative reduced can be identified. In fact, the set of dual values obtained during the last column generation iterations can be useful to perform effective variable fixing tests; moreover, the structure of quasi-optimal LMP solutions can be exploited by the primal heuristic to obtain tight bounds.

Experimental results show that this termination criterion reduces the computation time and the number of column generation iterations in non-root nodes. We made a comparison between the performances of our algorithm with and without the use of the Lagrangean lower bound on the set of instances with $N = 50$ of the four classes and we observed a reduction of about 8.5% in the overall CPU time required and a reduction of about 2.5% in the average number of column generation iterations needed in each non-root node of the search tree.

Subgradient optimization. It is also possible to better exploit the relationship between column generation and Lagrangean relaxation outlined above to improve the dual variables via subgradient optimization [47] after each column generation iteration. Starting with the current optimal values of the dual variables $\boldsymbol{\lambda^t}$, 100 subgradient iterations are executed. The step parameter is initialized to 2 and halved after every 10 iterations in which $\omega_{LR}^t \leq \omega_{LR}^{t-1}$, that is the current lower bound has not been improved with respect to the previous iteration.

At each subgradient iteration we also compute a primal solution using a Lagrangean heuristic: medians are chosen according to the values of the Lagrangean penalties; when partitioning constraints are not violated, we use the same assignments which appear in the solution of the RLMP; the allocation of the other vertices is done as in the MTHG algorithm, with desirability coefficients $f_{ij} = -d_{ij}$; then a local search step is performed, as proposed by Mulvey and Beck [78]: after every primal bound evaluation, the optimal median for every cluster is re-computed and a new primal bound is evaluated with the new set of medians; this process is iterated until no more changes in the medians occur or a limit of 5 iterations is reached.

This combination of column generation with subgradient optimization yielded significant improvements at the root node, in terms of reduced number of column generation iterations, reduced computational time to achieve convergence, increased number of variables fixed and quality of the primal bound. However, this improvement did not reduce neither the size of the search tree nor the time required by the algorithm to complete the enumeration. Furthermore, this method was too time-consuming to be used at each node of the search tree.

Variable fixing. Given a solution of the Lagrangean relaxation LR, let ω_{LR} be its value and let $j^{WI} \in \text{argmin}_{j \in \mathcal{M}^{LR}}\{\tau_j\}$ be the vertex with minimum τ_j value which is a median and $j^{BO} \in \text{argmax}_{j \notin \mathcal{M}^{LR}}\{\tau_j\}$ be the vertex with maximum τ_j value which is not a median (WI stands for "worst in", BO for "best out"). Let also

v^* be a primal bound. If it does exist $j \in \mathcal{M}^{LR}$ such that $\lceil \omega_{LR} + \tau_j - \tau_{j^{BO}} \rceil \geq v^*$, then y_j can be fixed to 1. Analogously, if it does exist $j \notin \mathcal{M}^{LR}$ such that $\lceil \omega_{LR} - \tau_j + \tau_{j^{WI}} \rceil \geq v^*$, then y_j can be fixed to 0 (this also implies $x_{ij} = 0 \; \forall i \in \mathcal{N}$).

Once the τ_j values have been computed, this variable fixing step takes $O(M)$ time and it may reduce the problem size considerably. In our experiments variable fixing was done at each iteration of the subgradient optimization algorithm at the root node, and only at the end of column generation at the other nodes in the search tree.

The effectiveness of the variable fixing test at the root node can be appreciated from the computational results reported in tables 6.3 to 6.6 in columns "fix.med.".

6.3.5 Pricing algorithm

In the pricing step we solve binary knapsack problem instances to optimality by a modified version of Pisinger's MINKNAP algorithm [87], that combines dynamic programming with bounding and reduction techniques to dynamically adjust the core. Yet MINKNAP was devised for knapsack problems with integer coefficients, while in our pricing subproblems the dual variables, as well as the multipliers in Lagrangean relaxation, can be fractional. Instead of scaling the coefficients, that implies the computation of the determinant of a matrix and can be very time-consuming, we relaxed the bounding test so that the optimal solution computed by the modified algorithm may differ from optimality by at most $N_{KP}\varepsilon$, where N_{KP} is the number of variables left outside the core and ε is a very small positive parameter. Although this technique gives slightly worse dual bounds, we found that it has a clear computational advantage compared to the scaling approach. In particular we considered values of ε in the range between 10^{-6} and 10^{-12}: for values higher than 10^{-6} the approximation is not tight; for values lower than 10^{-12} numerical problems may arise; in between these values we obtained tight approximations, and we observed that the computation time does not change significantly.

In general it is also possible to generate columns with negative reduced cost by solving the pricing problem approximately. This is usually done when the pricing problem is difficult to solve to optimality. However our computational experience showed that this is not worth doing in this case, since the time required to solve the binary knapsack is negligible compared with that needed to solve the LMP.

Column generation can be also speeded up by multiple pricing: instead of inserting into the RLMP only the optimal column for each candidate median, if any is found with negative reduced cost, it is worth adding more (suboptimal) columns to enlarge the search space for the linear programming algorithm. This is particularly useful at the root node, when the column pool is still empty and the set of available columns may be small.

To this purpose we exploit the subgradient optimization algorithm and we insert into the RLMP the set of columns corresponding to each solution of the LR for which the Lagrangean lower bound improves upon the best incumbent Lagrangean lower bound.

Another method we devised to obtain promising columns consists of applying local exchanges to the columns found at each column generation iteration. For each candidate median, we consider the optimal cluster, that is the cluster described by the column with the minimum reduced cost, and we generate all feasible clusters obtained by the exchange of one vertex in the cluster with one vertex outside the cluster. All columns generated in this way are added to the RLMP, provided that their reduced cost is negative and that it is no more than 10% far away from the optimal one.

In table 6.2, we present the computational results when (a) multiple pricing is not used, (b) multiple columns are generated via local exchanges, and (c) multiple columns are generated via subgradient optimization. For each policy we report the number of iterations of the column generation algorithm (CGRoot), the number of columns generated (ColsRoot) and the time spent (TimeRoot) at the root node. In the same way for the search tree nodes we report the average number of column generation iterations (CGTree), the number of columns generated (ColsTree) and the time spent to complete the enumeration process (TimeTree). Finally we report the number of nodes explored (Eval. Nodes).

In case (c) the number of column generation iterations and the time spent at the root node are inferior. Moreover, the CPU time reported in case (c) includes the time spent for the Lagrangean heuristic and the variable fixing routine. However in case (c) we often observed an increase in the computation time required to complete the overall enumeration process. This can be explained by at least two reasons: first, when the optimum of the LMP is degenerate, the solution obtained in case (c) has slightly more fractional variables; second, when the algorithm generates less columns at the root node, the pool is scarcely useful and more columns must be generated later during the exploration of other nodes.

6.3.6 Experimental results

We report in tables 6.3, 6.4, 6.5 and 6.6 detailed results on the computational behavior of the branch-and-price algorithm when subgradient-based multiple pricing is used. These tables, one for each class of instances, are composed by two horizontal blocks: the first block contains experimental results for the root node, while the second block refers to the whole search tree.

For the root node we report the number of column generation iterations needed to reach optimality (CG it.), the number of columns generated (cols), the best upper bound found (UB), the value of the LMP relaxation (LB), the gap between

(a) No multiple pricing

N	p	CGRoot	ColsRoot	TimeRoot(s)	CGTree	ColsTree	TimeTree(s)	Eval. nodes
50	5	107.3	4566.0	1.38	4.27	51.02	207.05	1304.6
	12	77.7	3589.4	0.43	3.76	58.01	93.10	2770.0
	16	73.7	3422.1	0.33	3.64	60.77	19.17	13707.7
	20	71.4	3331.3	0.30	2.84	47.69	35.92	1727.0
100	10	168.8	15166.6	12.83	10.98	259.55	214.56	1336.5
	25	137.9	13099.0	4.68	6.59	222.57	622.46	19984.2
	33	132.6	12680.8	3.93	4.83	175.83	505.28	21048.5
	40	130.4	12527.7	3.46	3.50	128.70	619.98	20357.2
Average		112.48	8547.86	3.42	5.05	125.52	289.69	10279.46

(b) Multiple pricing via local search

N	p	CGRoot	ColsRoot	TimeRoot(s)	CGTree	ColsTree	TimeTree(s)	Eval. nodes
50	5	80.7	25458.8	10.04	5.48	67.90	228.75	1288.0
	12	64.5	10921.6	1.30	4.71	75.25	83.31	2318.0
	16	62.9	8684.4	0.79	3.61	62.66	16.46	13230.3
	20	61.4	7067.6	0.49	2.55	44.80	50.91	2324.6
100	10	133.5	129155.0	146.71	10.08	237.93	418.48	1187.3
	25	117.3	50237.6	15.05	6.35	214.95	510.33	18263.3
	33	116.0	39770.0	9.31	5.31	194.48	328.28	18751.4
	40	114.3	33000.8	6.32	3.71	138.41	436.83	17172.5
Average		93.83	38036.98	23.75	5.23	129.55	259.17	9316.93

(c) Multiple pricing via subgradient optimization

N	p	CGRoot	ColsRoot	TimeRoot(s)	CGTree	ColsTree	TimeTree(s)	Eval. nodes
50	5	8.9	3526.7	0.77	12.65	109.90	189.11	1204.6
	12	7.0	2139.1	0.55	6.93	95.97	90.45	2495.6
	16	6.3	2274.6	0.51	5.34	80.00	15.46	13450.5
	20	5.4	2341.6	0.51	4.06	62.68	49.40	2188.2
100	10	12.8	8299.3	3.53	20.76	416.74	543.60	1248.7
	25	7.2	4887.1	2.46	8.16	258.21	115.67	18858.7
	33	6.6	5222.8	2.33	6.56	235.11	467.95	18065.1
	40	5.3	4612.3	1.83	5.56	199.96	795.43	17850.5
Average		7.44	4162.94	1.56	8.75	182.32	283.39	9420.24

Table 6.2: Comparison between different multiple pricing methods

the upper and lower bounds, that is $\frac{UB-LB}{LB}$ (gap). Moreover, we report the gap between the upper bound at the root node and the best solution found during the exploration of the search tree (UB gap $= \frac{UB-F.UB}{F.UB}$) and the gap between the initial LMP relaxation and the best solution found (LB gap $= \frac{F.UB-LB}{F.UB}$). The former evaluates the quality of the heuristic solution found by column generation, the latter the quality of the LMP relaxation. Finally we include the CPU time spent (time) and number of candidate medians fixed to 0 by the variable fixing tests (fix.med.) at the root node.

For the search tree we report the average number of iterations of the column generation algorithm (CG it.) and the average number of generated columns in each node of the search tree (cols), the final upper and lower bounds (F.UB and F.LB), the approximation obtained (gap), the CPU time spent in the optimization (the tests for which computation exceeded time limit are marked with a dash), the average number of variable fixing tests succeeded in each node of the search tree (avg.fix.med.) and the number of nodes evaluated in the search tree (ev. nodes).

Instance			Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time(s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time(s)	avg fix.med.	ev. nodes
50	5	cpmp01	13	2706	713	705	1.13%	0.00%	1.12%	0.60	28	16.00	139.84	713	713	0.00%	4.58	2.73	33
		cpmp02	3	896	740	740	0.00%	0.00%	0.00%	0.15	0	0.00	0.00	740	740	0.00%	0.16	0.00	1
		cpmp03	13	2755	751	749	0.27%	0.00%	0.27%	0.66	40	40.25	254.25	751	751	0.00%	2.43	0.00	5
		cpmp04	4	1571	651	651	0.00%	0.00%	0.00%	0.28	0	0.00	0.00	651	651	0.00%	0.31	0.00	1
		cpmp05	5	1441	664	664	0.00%	0.00%	0.00%	0.54	0	0.00	0.00	664	664	0.00%	0.56	0.00	1
		cpmp06	3	1598	778	778	0.00%	0.00%	0.00%	0.30	0	0.00	0.00	778	778	0.00%	0.33	0.00	1
		cpmp07	10	2650	787	779	1.03%	0.00%	1.11%	0.59	22	14.00	167.00	787	787	0.00%	7.13	3.06	37
		cpmp08	11	2835	820	772	6.22%	0.00%	5.89%	0.86	2	20.63	210.05	820	820	0.00%	1904.73	3.61	11965
		cpmp09	10	2575	715	713	0.28%	0.00%	0.34%	0.51	30	10.00	74.00	715	715	0.00%	1.00	4.00	5
		cpmp10	9	1829	829	818	1.34%	0.00%	1.34%	0.78	17	10.57	97.99	829	829	0.00%	7.96	3.26	109
		Average	8.1	2085.6			1.03%	0.00%	1.01%	0.53	13.9	11.14	94.31			0.00%	192.92	1.67	1215.8
100	10	cpmp11	11	5573	1007	1002	0.50%	0.10%	0.49%	2.16	40	19.50	414.58	1006	1006	0.00%	19.08	10.08	25
		cpmp12	11	4896	969	959	1.04%	0.31%	0.78%	1.73	20	17.04	358.63	966	966	0.00%	93.51	7.13	211
		cpmp13	13	5568	1026	1022	0.39%	0.00%	0.43%	2.94	66	38.86	809.07	1026	1026	0.00%	30.46	5.00	15
		cpmp14	11	4880	985	972	1.34%	0.31%	1.04%	2.15	24	17.06	351.92	982	982	0.00%	232.13	6.47	425
		cpmp15	12	5623	1092	1081	1.02%	0.09%	0.97%	2.55	28	17.86	311.71	1091	1091	0.00%	407.73	5.21	721
		cpmp16	8	3940	955	952	0.32%	0.10%	0.28%	1.55	60	16.70	266.50	954	954	0.00%	5.70	4.14	11
		cpmp17	13	4603	1034	1026	0.78%	0.00%	0.84%	2.26	33	17.06	249.75	1034	1034	0.00%	113.91	5.03	283
		cpmp18	10	5053	1046	1032	1.36%	0.29%	1.06%	1.96	25	21.25	476.07	1043	1043	0.00%	877.95	6.35	1181
		cpmp19	13	5567	1036	1027	0.88%	0.48%	0.46%	2.66	27	13.57	329.41	1031	1031	0.00%	18.03	5.18	45
		cpmp20	11	4906	1022	974	4.93%	1.59%	3.22%	2.83	0	19.04	474.55	1006	993	1.31%	–	8.87	9128
		Average	11.3	5060.9			1.25%	0.33%	0.96%	2.28	32.3	19.79	404.22			0.13%	199.83	6.35	1204.5
150	15	cpmp21	10	6576	1294	1282	0.94%	0.47%	0.48%	4.02	26	14.97	526.78	1288	1288	0.00%	81.86	8.36	105
		cpmp22	12	8969	1256	1248	0.64%	0.00%	0.65%	6.14	39	20.13	505.33	1256	1255	0.08%	–	6.00	3182
		cpmp23	11	7898	1279	1278	0.08%	0.00%	0.09%	5.29	90	8.50	230.00	1279	1279	0.00%	6.89	0.00	3
		cpmp24	11	6461	1220	1219	0.08%	0.00%	0.13%	5.22	100	22.50	536.00	1220	1220	0.00%	8.35	0.00	3
		cpmp25	10	6135	1193	1189	0.34%	0.00%	0.35%	5.15	68	29.81	918.35	1193	1193	0.00%	125.71	6.60	53
		cpmp26	11	7240	1269	1259	0.79%	0.40%	0.41%	5.67	20	16.63	641.36	1264	1264	0.00%	184.92	16.11	121
		cpmp27	10	7838	1330	1312	1.37%	0.53%	0.85%	5.54	15	17.58	599.31	1323	1320	0.23%	–	12.26	3139
		cpmp28	11	7200	1237	1231	0.49%	0.32%	0.20%	4.65	45	26.70	1030.20	1233	1233	0.00%	20.72	19.75	11
		cpmp29	5	5850	1219	1219	0.00%	0.00%	0.00%	7.36	0	0.00	0.00	1219	1219	0.00%	7.56	0.00	1
		cpmp30	13	8536	1205	1201	0.33%	0.33%	0.06%	5.41	53	61.50	4413.75	1201	1201	0.00%	34.76	12.75	5
		Average	10.4	7270.3			0.51%	0.20%	0.32%	5.45	45.6	21.83	940.11			0.03%	58.85	8.18	662.3
200	20	cpmp31	11	9913	1379	1373	0.44%	0.07%	0.43%	10.20	49	31.01	1847.13	1378	1378	0.00%	1003.81	13.10	223
		cpmp32	12	8687	1429	1410	1.35%	0.00%	1.34%	12.48	12	24.16	1803.95	1429	1419	0.70%	–	7.82	1149
		cpmp33	15	11154	1383	1362	1.54%	1.17%	0.38%	11.66	6	23.01	1659.16	1367	1367	0.00%	1420.90	21.10	557
		cpmp34	12	11092	1385	1375	0.73%	0.00%	0.75%	13.34	44	25.67	1046.43	1385	1383	0.14%	–	11.76	1425
		cpmp35	10	8917	1442	1431	0.77%	0.35%	0.44%	8.75	23	19.77	920.21	1437	1437	0.00%	2662.50	16.06	1265
		cpmp36	11	9638	1385	1379	0.44%	0.22%	0.27%	8.41	53	26.77	1611.16	1382	1382	0.00%	188.59	11.21	65
		cpmp37	11	10360	1458	1455	0.21%	0.00%	0.26%	7.90	84	32.94	1730.03	1458	1458	0.00%	109.22	14.73	33
		cpmp38	12	9379	1400	1373	1.97%	1.01%	0.99%	10.13	0	26.52	2195.10	1386	1381	0.36%	–	21.93	1100
		cpmp39	11	8568	1389	1370	1.39%	1.09%	0.31%	7.98	9	26.45	1998.51	1374	1374	0.00%	222.67	26.75	81
		cpmp40	10	8967	1432	1414	1.27%	1.13%	0.20%	8.50	3	18.79	1351.40	1416	1416	0.00%	167.56	19.45	87
		Average	11.5	9667.5			1.01%	0.50%	0.54%	9.93	28.3	25.51	1616.31			0.12%	825.04	16.39	598.5
Overall average			10.33	6021.08			0.95%	0.26%	0.71%	4.55		19.57	763.74			0.07%	319.16		920.28

Table 6.3: Branch-and-price with subgradient-based multiple pricing - Class α

Instance			Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time(s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time(s)	avg fix.med.	ev. nodes
50	12	cpmp01	4	1285	387	374	3.48%	1.04%	2.52%	0.15	1	7.05	94.36	383	383	0.00%	5.75	5.25	165
		cpmp02	6	1361	420	409	2.69%	1.94%	0.92%	0.26	1	7.75	142.25	412	412	0.00%	1.86	2.57	37
		cpmp03	5	1201	405	399	1.50%	0.00%	1.67%	0.24	3	6.76	90.20	405	405	0.00%	4.38	4.46	107
		cpmp04	6	1618	385	365	5.48%	0.26%	4.97%	0.33	0	8.08	102.44	384	384	0.00%	308.15	4.21	8075
		cpmp05	4	880	429	419	2.39%	0.00%	2.38%	0.21	4	6.47	81.06	429	429	0.00%	11.19	4.61	359
		cpmp06	5	1302	485	467	3.85%	0.62%	3.28%	0.28	1	7.97	118.55	482	482	0.00%	73.69	3.70	2633
		cpmp07	5	1235	445	425	4.71%	0.00%	4.64%	0.33	0	7.74	97.50	445	445	0.00%	318.60	3.61	8431
		cpmp08	6	1199	407	393	3.56%	0.99%	2.55%	0.34	2	6.70	100.49	403	403	0.00%	18.82	4.09	657
		cpmp09	5	1227	452	423	6.86%	3.67%	3.11%	0.41	0	6.83	94.47	436	436	0.00%	23.68	5.42	623
		cpmp10	5	1309	466	443	5.19%	1.08%	4.09%	0.41	1	6.51	93.69	461	461	0.00%	113.35	4.44	3607
		Average	5.1	1261.7			3.97%	0.96%	3.01%	0.30	1.3	7.19	101.50			0.00%	87.95	4.24	2469.4
100	25	cpmp11	6	3101	549	529	3.78%	0.92%	2.77%	1.01	0	7.73	235.57	544	544	0.00%	2963.49	8.58	25757
		cpmp12	5	2190	508	496	2.42%	0.79%	1.59%	0.80	2	6.94	169.11	504	504	0.00%	99.66	8.80	1017
		cpmp13	5	2392	569	535	6.36%	2.15%	4.12%	0.76	0	8.56	337.38	557	546	2.01%	–	2.25	32375
		cpmp14	5	2938	556	530	4.91%	2.21%	2.67%	0.79	0	9.81	287.92	544	541	0.55%	–	6.91	28074
		cpmp15	5	2611	586	573	2.27%	0.51%	1.77%	0.90	3	7.56	196.19	583	583	0.00%	231.80	9.04	1707
		cpmp16	6	2566	543	521	4.22%	0.93%	3.32%	1.08	0	8.25	299.15	538	530	1.51%	–	3.20	36101
		cpmp17	5	2352	551	536	2.80%	1.66%	1.14%	0.96	1	6.37	154.73	542	542	0.00%	6.60	9.17	53
		cpmp18	8	4408	508	501	1.40%	0.00%	1.38%	1.33	10	10.89	342.38	508	508	0.00%	47.78	7.88	179
		cpmp19	5	2879	562	531	5.84%	2.00%	3.69%	0.85	0	8.60	320.10	551	547	0.73%	–	7.75	31511
		cpmp20	4	1930	581	523	11.09%	1.57%	8.63%	1.04	0	7.98	300.49	572	537	6.52%	–	0.06	32135
		Average	5.4	2736.7			4.51%	1.27%	3.11%	0.95	1.6	8.27	264.30			1.13%	669.87	6.36	18890.9
150	37	cpmp21	6	4793	682	675	1.04%	0.15%	0.98%	2.01	4	10.36	380.63	681	681	0.00%	838.88	11.00	2685
		cpmp22	5	2703	680	646	5.26%	1.80%	3.40%	2.21	0	8.43	463.76	668	654	2.14%	–	0.94	13447
		cpmp23	7	4599	682	643	6.07%	1.79%	4.17%	2.88	0	8.81	486.25	670	652	2.76%	–	0.31	12794
		cpmp24	6	4884	597	587	1.70%	0.51%	1.21%	2.01	2	8.83	361.28	594	594	0.00%	2365.36	11.56	8527
		cpmp25	6	3893	636	628	1.27%	1.11%	0.29%	1.63	0	6.41	338.03	629	629	0.00%	24.07	7.50	103
		cpmp26	6	4953	680	647	5.10%	4.13%	1.06%	1.86	0	8.66	392.74	653	653	0.00%	188.63	13.15	599
		cpmp27	6	4138	771	714	7.98%	2.12%	5.51%	2.70	0	7.93	435.58	755	723	4.43%	–	0.00	13355
		cpmp28	5	3785	654	633	3.32%	1.55%	1.84%	1.58	0	8.75	472.12	644	641	0.47%	–	10.97	14962
		cpmp29	6	5180	670	641	4.52%	3.24%	1.29%	1.98	0	7.09	309.75	649	649	0.00%	472.24	17.00	2135
		cpmp30	7	5092	634	619	2.42%	0.48%	1.90%	2.24	0	8.77	418.25	631	628	0.48%	–	9.79	17376
		Average	6.0	4402.0			3.87%	1.69%	2.17%	2.11	0.6	8.40	405.84			1.03%	777.84	8.22	8598.3
200	50	cpmp31	5	5421	748	712	5.06%	1.63%	3.29%	3.58	0	7.83	550.64	736	718	2.51%	–	0.10	7376
		cpmp32	5	4647	908	800	13.50%	3.42%	8.97%	4.42	0	6.34	346.10	878	806	8.93%	–	0.00	6050
		cpmp33	6	5397	726	694	4.61%	0.14%	4.32%	4.61	0	8.82	642.66	725	701	3.42%	–	0.00	7024
		cpmp34	6	4867	854	779	9.63%	0.00%	8.81%	5.64	0	7.40	402.77	854	785	8.79%	–	0.00	6187
		cpmp35	6	6955	763	728	4.81%	0.66%	4.07%	4.26	0	7.79	539.26	758	733	3.41%	–	0.00	7504
		cpmp36	7	7652	708	693	2.16%	0.71%	1.53%	4.96	0	8.13	586.80	703	697	0.86%	–	3.77	7682
		cpmp37	5	5700	785	744	5.51%	3.97%	1.54%	2.89	0	8.72	655.21	755	749	0.80%	–	3.77	7898
		cpmp38	5	4365	775	730	6.16%	0.00%	5.81%	4.20	0	7.89	531.62	775	735	5.44%	–	0.00	6134
		cpmp39	7	7417	736	713	3.23%	1.80%	1.44%	4.10	0	8.87	595.13	723	720	0.42%	–	13.79	6916
		cpmp40	6	6377	796	738	7.86%	0.38%	6.99%	5.74	0	7.83	454.32	793	745	6.44%	–	0.00	6192
		Average	5.8	5879.8			6.25%	1.27%	4.68%	4.44	0.0	7.96	530.45			4.10%	–	2.14	6896.3
Overall average			5.58	3570.05			4.65%	1.30%	3.24%	1.95		7.96	325.52			1.57%	511.88		9213.73

Table 6.4: Branch-and-price with subgradient-based multiple pricing - Class β

Instance			Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time(s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time(s)	avg fix.med.	ev. nodes
50	16	cpmp01	4	1537	305	296	3.04%	2.35%	0.67%	0.16	2	4.67	82.50	298	298	0.00%	0.42	1.00	7
		cpmp02	3	814	338	328	3.05%	0.60%	2.38%	0.13	2	5.19	85.08	336	336	0.00%	3.51	2.92	143
		cpmp03	4	1412	317	309	2.59%	0.96%	1.59%	0.15	0	4.80	79.50	314	314	0.00%	0.58	4.40	11
		cpmp04	4	1280	303	297	2.02%	0.00%	2.29%	0.21	5	4.93	64.51	303	303	0.00%	2.23	2.91	71
		cpmp05	5	1555	351	346	1.45%	0.00%	1.42%	0.25	2	5.06	79.44	351	351	0.00%	0.94	4.38	19
		cpmp06	4	1019	403	387	4.13%	3.33%	0.90%	0.25	0	5.25	93.19	390	390	0.00%	0.75	1.00	17
		cpmp07	5	1747	362	357	1.40%	0.28%	1.11%	0.30	5	4.36	69.29	361	361	0.00%	3.19	3.17	95
		cpmp08	5	1237	363	323	12.38%	2.54%	8.86%	0.34	0	7.64	146.61	354	349	1.43%	–	4.87	127066
		cpmp09	4	1010	387	366	5.74%	3.75%	1.88%	0.27	0	4.16	62.05	373	373	0.00%	9.34	4.08	407
		cpmp10	4	1042	399	376	6.12%	2.31%	3.79%	0.32	0	5.93	84.96	390	390	0.00%	143.78	2.98	5137
		Average	4.2	1265.3			4.19%	1.61%	2.49%	0.24	1.6	5.20	84.71			0.14%	18.30	3.17	13297.3
100	33	cpmp11	4	2165	420	406	3.45%	1.45%	1.97%	0.56	0	6.27	217.97	414	414	0.00%	97.42	6.72	1053
		cpmp12	5	4854	402	375	7.20%	2.81%	4.16%	0.75	0	8.11	303.72	391	387	1.03%	–	6.11	34995
		cpmp13	5	4169	446	441	1.13%	0.00%	1.23%	0.75	3	6.67	209.77	446	446	0.00%	30.25	6.09	203
		cpmp14	4	1672	455	434	4.84%	1.79%	2.91%	0.70	0	7.19	270.52	447	443	0.90%	–	6.19	39581
		cpmp15	4	2686	517	470	10.00%	9.07%	0.98%	0.68	0	6.29	230.82	474	474	0.00%	135.43	2.82	1317
		cpmp16	4	2483	454	431	5.34%	0.44%	4.77%	0.72	0	6.49	265.22	452	440	2.73%	–	0.77	41818
		cpmp17	5	3931	447	424	5.42%	3.71%	1.78%	0.81	0	6.46	210.82	431	431	0.00%	120.12	6.06	855
		cpmp18	5	2211	462	431	7.19%	0.00%	6.78%	1.06	0	6.64	269.58	462	442	4.52%	–	0.37	39287
		cpmp19	5	3675	471	431	9.28%	5.84%	3.22%	0.87	0	6.39	189.70	445	445	0.00%	1399.44	7.08	12989
		cpmp20	4	2772	470	451	4.21%	2.17%	1.96%	0.88	0	5.91	176.56	460	460	0.00%	847.31	7.81	9321
		Average	4.5	3061.8			5.81%	2.73%	2.98%	0.78	0.3	6.64	234.47			0.92%	438.33	5.00	18141.9
150	50	cpmp21	6	4249	612	581	5.34%	0.00%	5.13%	3.46	0	6.45	363.75	612	588	4.08%	–	0.00	15748
		cpmp22	4	2694	590	540	9.26%	2.08%	6.60%	2.06	0	7.05	412.41	578	548	5.47%	–	0.00	15096
		cpmp23	5	4655	590	551	7.08%	3.87%	3.02%	2.52	0	7.31	448.29	568	559	1.61%	–	1.53	15158
		cpmp24	4	2917	522	496	5.24%	3.37%	1.95%	1.54	0	7.14	414.21	505	503	0.40%	–	8.08	15747
		cpmp25	4	3652	512	486	5.35%	4.92%	0.58%	1.21	0	6.64	439.24	488	488	0.00%	13.09	1.42	59
		cpmp26	5	4893	547	526	3.99%	1.30%	2.65%	1.67	0	6.66	383.68	540	537	0.56%	–	9.27	17439
		cpmp27	5	5124	610	568	7.39%	5.35%	2.01%	2.17	0	7.84	398.79	579	578	0.17%	–	10.40	14258
		cpmp28	4	2385	511	499	2.40%	1.59%	0.80%	1.16	0	4.87	247.27	503	503	0.00%	72.69	4.76	483
		cpmp29	5	5378	554	530	4.53%	0.00%	4.46%	1.77	0	6.52	376.30	554	535	3.55%	–	0.03	17533
		cpmp30	5	4010	521	488	6.76%	3.17%	3.50%	1.87	0	7.02	399.80	505	495	2.02%	–	1.10	18879
		Average	4.7	3995.7			5.73%	2.56%	3.07%	1.94	0.0	6.75	388.37			1.79%	42.89	3.66	13040.0
200	66	cpmp31	5	5247	579	572	1.22%	0.70%	0.68%	3.27	0	6.01	413.83	575	574	0.17%	–	8.81	11898
		cpmp32	4	5238	833	700	19.00%	7.07%	10.13%	3.67	0	5.08	282.11	778	705	10.35%	–	0.00	8084
		cpmp33	4	4901	654	600	9.00%	0.00%	8.39%	4.31	0	6.07	383.52	654	605	8.10%	–	0.00	7349
		cpmp34	6	5678	749	684	9.50%	0.00%	8.74%	6.74	0	6.71	489.37	749	691	8.39%	–	0.00	6745
		cpmp35	6	5718	631	593	6.41%	1.94%	4.29%	5.25	0	6.67	540.47	619	598	3.51%	–	0.00	8535
		cpmp36	5	5341	608	570	6.67%	0.00%	6.33%	4.76	0	6.58	456.59	608	574	5.92%	–	0.00	7025
		cpmp37	4	5787	655	617	6.16%	0.00%	5.88%	3.00	0	6.28	507.32	655	621	5.48%	–	0.00	6898
		cpmp38	5	4592	633	596	6.21%	4.11%	2.08%	3.86	0	6.53	524.49	608	601	1.16%	–	2.34	9655
		cpmp39	4	5743	609	574	6.10%	2.35%	3.60%	3.58	0	6.43	523.50	595	578	2.94%	–	0.00	7995
		cpmp40	4	3513	657	608	8.06%	1.55%	6.04%	4.11	0	6.44	469.29	647	613	5.55%	–	0.00	7761
		Average	4.7	5175.8			7.83%	1.77%	5.62%	4.25	0.0	6.28	459.05			5.16%	–	1.11	8194.5
Overall average			4.53	3374.65			5.89%	2.17%	3.54%	1.80		6.22	291.65			2.00%	166.51		13168.43

Table 6.5: Branch-and-price with subgradient-based multiple pricing - Class γ

Instance			Root									Search tree							
N	p		CG it.	cols	UB	LB	gap	UB gap	LB gap	time(s)	fix.med.	CG it.	cols	F.UB	F.LB	gap	time(s)	avg fix.med.	ev. nodes
50	20	cpmp01	3	1066	266	259	2.70%	0.00%	2.74%	0.13	2	4.91	74.95	266	266	0.00%	1.69	3.33	57
		cpmp02	4	2107	300	293	2.39%	0.67%	1.85%	0.31	4	4.44	67.60	298	298	0.00%	3.43	1.91	79
		cpmp03	3	1000	352	307	14.66%	13.18%	1.29%	0.21	0	4.06	71.38	311	311	0.00%	0.66	0.75	17
		cpmp04	3	784	277	276	0.36%	0.00%	0.54%	0.15	6	2.69	40.75	277	277	0.00%	0.44	0.43	17
		cpmp05	4	1550	360	355	1.41%	1.12%	0.42%	0.38	1	4.00	54.00	356	356	0.00%	0.57	3.67	5
		cpmp06	4	1400	370	367	0.82%	0.00%	0.81%	0.26	6	3.00	8.50	370	370	0.00%	0.35	0.00	3
		cpmp07	4	1356	359	357	0.56%	0.28%	0.28%	0.36	6	3.83	62.83	358	358	0.00%	0.60	3.60	7
		cpmp08	4	914	322	298	8.05%	3.21%	4.78%	0.33	0	4.44	73.65	312	312	0.00%	44.90	2.90	2025
		cpmp09	3	739	417	404	3.22%	1.21%	2.09%	0.29	0	3.86	64.30	412	412	0.00%	11.65	1.90	539
		cpmp10	3	920	502	442	13.57%	9.61%	3.68%	0.39	0	4.48	77.35	458	458	0.00%	426.68	2.65	19429
		Average	3.5	1183.6			4.77%	2.93%	1.85%	0.28	2.5	3.97	59.53			0.00%	49.10	2.11	2217.8
100	40	cpmp11	4	3381	446	406	9.85%	7.47%	2.21%	0.89	0	6.45	249.48	415	413	0.48%	–	5.89	38912
		cpmp12	4	2035	424	365	16.16%	11.58%	4.14%	1.04	0	5.93	237.71	380	373	1.88%	–	1.88	42328
		cpmp13	4	3955	424	405	4.69%	2.91%	1.74%	0.74	0	4.54	147.86	412	412	0.00%	430.11	4.05	4449
		cpmp14	4	2230	474	413	14.77%	12.59%	1.97%	0.92	0	5.29	172.20	421	421	0.00%	352.47	3.72	3731
		cpmp15	3	1924	526	489	7.57%	6.05%	1.44%	0.68	0	5.50	178.90	496	496	0.00%	737.53	5.60	9075
		cpmp16	4	3423	435	425	2.35%	1.64%	0.86%	0.91	0	4.26	147.66	428	428	0.00%	95.71	2.95	1205
		cpmp17	4	1848	475	430	10.47%	7.95%	2.27%	0.95	0	5.85	190.92	440	440	0.00%	1588.28	5.14	18091
		cpmp18	5	3857	464	434	6.91%	3.11%	3.69%	1.13	0	5.75	191.39	450	450	0.00%	666.00	4.82	6675
		cpmp19	5	2901	466	440	5.91%	3.56%	2.43%	1.57	0	5.25	197.79	450	450	0.00%	3587.27	4.90	40217
		cpmp20	5	5076	523	476	9.87%	7.61%	2.19%	1.81	0	5.19	180.20	486	486	0.00%	533.02	4.71	4773
		Average	4.2	3063.0			8.86%	6.45%	2.30%	1.06	0.0	5.40	189.41			0.24%	998.80	4.36	16945.6
150	60	cpmp21	6	4780	591	545	8.44%	7.07%	1.28%	3.23	0	5.81	367.01	552	552	0.00%	333.66	2.18	1481
		cpmp22	4	4519	659	588	12.07%	9.65%	2.21%	2.46	0	6.06	385.59	601	595	1.01%	–	2.40	18434
		cpmp23	3	3160	597	540	10.56%	4.01%	5.99%	1.59	0	5.68	353.92	574	548	4.74%	–	0.00	17100
		cpmp24	4	2386	506	475	6.53%	1.61%	4.80%	2.01	0	5.91	367.48	498	480	3.75%	–	0.02	17603
		cpmp25	4	4910	478	428	11.68%	9.63%	1.90%	1.77	0	5.91	321.61	436	434	0.46%	–	7.28	21154
		cpmp26	4	2433	534	506	5.53%	3.89%	1.65%	2.70	0	5.36	329.25	514	511	0.59%	–	4.34	16901
		cpmp27	5	2838	772	715	7.97%	0.39%	7.09%	3.47	0	5.74	363.25	769	722	6.51%	–	0.00	14311
		cpmp28	4	4844	485	463	4.75%	2.75%	2.03%	1.58	0	5.39	292.07	472	469	0.64%	–	5.39	23304
		cpmp29	4	4988	524	488	7.38%	6.07%	1.21%	1.30	0	5.24	303.62	494	494	0.00%	1976.75	6.63	11337
		cpmp30	5	7058	461	435	5.98%	3.83%	2.06%	1.96	0	5.84	335.76	444	441	0.68%	–	6.93	19715
		Average	4.3	4191.6			8.09%	4.89%	3.02%	2.21	0.0	5.69	341.95			1.84%	1155.21	3.52	16134.0
200	80	cpmp31	4	7505	567	528	7.39%	3.66%	3.50%	3.76	0	5.82	397.49	547	533	2.63%	–	0.04	7983
		cpmp32	4	3796	926	781	18.57%	9.20%	7.96%	5.80	0	5.94	500.34	848	788	7.61%	–	0.00	7657
		cpmp33	4	4155	575	548	4.93%	1.05%	3.72%	3.86	0	5.22	403.86	569	551	3.27%	–	0.02	8852
		cpmp34	6	6058	1043	835	24.91%	20.02%	4.01%	9.57	0	6.14	513.67	869	838	3.70%	–	0.00	6097
		cpmp35	4	4175	588	542	8.49%	3.34%	4.75%	4.57	0	5.44	453.69	569	545	4.40%	–	0.00	8076
		cpmp36	4	6777	581	538	7.99%	0.00%	7.42%	3.53	0	5.66	436.61	581	543	7.00%	–	0.00	7576
		cpmp37	5	7722	634	580	9.31%	3.59%	5.28%	5.64	0	5.65	437.58	612	584	4.79%	–	0.00	6686
		cpmp38	5	6375	635	587	8.18%	7.26%	0.85%	5.33	0	5.96	489.10	592	590	0.34%	–	10.51	8422
		cpmp39	4	3591	578	528	9.47%	0.00%	8.66%	4.95	0	5.20	366.94	578	532	8.65%	–	0.00	5775
		cpmp40	4	4433	632	571	10.68%	3.61%	6.40%	3.91	0	5.49	410.60	610	575	6.09%	–	0.00	6401
		Average	4.4	5458.7			10.99%	5.17%	5.26%	5.09	0.0	5.65	440.99			4.85%	–	1.06	7352.5
Overall average			4.10	3474.23			8.18%	4.86%	3.11%	2.16		5.18	257.97			1.73%	734.37		10662.48

Table 6.6: Branch-and-price with subgradient-based multiple pricing - Class δ

The number of column generation iterations and the number of generated columns at the root node decrease as the ratio $\frac{p}{N}$ increases. This is especially true moving from class α to class β. We explained this phenomenon with the following observation: when the ratio is high, there are several medians with similar allocation pattern. The reduced cost of the corresponding columns is almost the same and many columns are inserted into the RLMP at each column generation iteration. On the opposite the performances of the primal heuristic worsen as the ratio increases, as it is harder to find an optimal set of medians. Instances of class γ are the hardest to solve for the branch-and-price algorithm: the average gap between the upper and lower bounds after one hour of computation and the number of unsolved instances are higher than for the other classes. A similar behavior has been observed for the uncapacitated version of the problem (see [21]). Finally we ran our algorithm without time limit. It was able to solve to optimality 15 more instances, and to reduce the average gap between the best known upper and lower bounds below 1% before running out of memory.

Large scale instances. We tried to use our algorithm on even larger instances: in order to obtain benchmark instances similar to those presented before, we used the instances pmed-38, pmed-39 and pmed-40 for the uncapacitated p-median problem from the OR Library, that involve 900 vertices, fixing the number of medians to 90. We generated random weights as described in the previous section, and set the capacity of each candidate median to 120. Our algorithm solved the root problem in less than 40 minutes, producing solutions whose primal-dual gap was less than 2.6%, 2.4% and 3.0%. No significant improvement was observed in these bounds after some hours of computation. However this was expected: no CPMP instance of this dimension has been solved so far, although larger instances have been solved for the uncapacitated version of the problem. This puts in evidence that the capacity constraints actually make the problem harder.

6.4 Benchmarks and experimental comparisons

6.4.1 Benchmark algorithms

General purpose solver. We solved the CPMP instances with CPLEX 6.5, using the formulation of the CPMP presented in Section 6.2, tightened with the inequalities $x_{ij} \leq y_j$. All the parameters were kept at the default values. These include automatic dynamic generation of clique, cover and GUB-cover inequalities, best-bound-first search strategy and a relative and absolute optimality tolerance of 0.01% and 10^{-6} respectively.

Lagrangean relaxation. Lagrangean relaxation has been successfully applied to many combinatorial optimization problems; hence it is a good benchmark for other methods. Pirkul's branch-and-bound algorithm for the capacitated concentrators location problem [86], based on the Lagrangean relaxation of the partitioning constraints, can be easily adapted to the CPMP. Also Baldacci et al. [6] compared their algorithm with an adaptation of Pirkul's algorithm. Following [86] and [6], we implemented a branch-and-bound algorithm based on the Lagrangean relaxation of the semi-assignment constraints presented in Section 6.3. The Lagrangean dual problem is solved by subgradient optimization. At most 300 subgradient iterations are executed at the root node, at most 50 iterations at the other nodes of the first level search tree (where branching is done on location variables) and at most 15 iterations at the nodes of the second level branching tree (where branching is done on assignment variables). Primal bounds are computed at each subgradient iteration by Pirkul's procedure Heur2.

We also implemented alternative and faster bounding techniques proposed in [86], namely evaluating the dual bound using the best Lagrangean penalties found at the predecessor node in the search tree, and solving the linear relaxation of the knapsack subproblems using a "good" set of multipliers.

More details on the adaptation of Pirkul's algorithm to the CPMP can be found in [14], where an alternative formulation is also discussed, consisting of the relaxation of constraints (6.1) and (6.3).

6.4.2 Algorithms comparison

In tables 6.7 and 6.8, for every class of instances the column "v^*(gap)" reports the optimal value if optimality was proven; otherwise it reports the gap between the best primal and dual bounds obtained. As indicated in Subsection 6.2.4, computation was halted after one hour or in case of memory overflow. If computation exceeded these resource limitations, the "time" column is marked with a dash and the type of resource exceeded is indicated in the "status" column. The last row of each block in the tables reports the average gap, the average computation time (neglecting the tests that exceeded resource limitations) and the number of problems solved to proven optimality.

From the examination of the gaps between the primal solution, the optimum and the lower bound at the root node, the branch-and-price algorithm shows a good behavior also when it is used as a heuristic. This is especially true for instances in class α, where the best solution found is on the average 0.26% from optimality after only 4.55 seconds. For all sizes considered CPLEX could solve to optimality more instances in class β than the other algorithms.

The Lagrangean approach shows a completely different behavior, since it is competitive even for large instances but only for small values of the $\frac{p}{N}$ ratio. When the number of medians increases, its performances significantly worsen. This is easy to explain: for given N, when p is higher the first level branching tree grows larger, since it is necessary to fix more location variables to reach a leaf node.

On the forty instances with $N = 200$, the primal solutions found by branch-and-price after one hour were consistently better than those found by any of the other algorithms considered. CPLEX was not able to find any feasible solution within the time limit for 11 of these 40 instances. For the remaining 29 instances, the average approximation error, computed with respect to the best known lower bound, was 2.64% for the branch-and-price algorithm, while the approximation error yielded by the Lagrangean relaxation algorithm was 7.57% and that given by CPLEX more than 40%.

6.4.3 The algorithm of Baldacci, Hadjicostantinou, Maniezzo and Mingozzi

In [6] Baldacci et al. proposed an algorithm (in the remainder we call it BHMM for short) that is able to prove optimality of the solution found or to provide a valid dual bound and therefore an *a posteriori* approximation guarantee.

BHMM adopts a three steps approach: a primal-dual bound gap is computed by a procedure H1, that relies on Lagrangean relaxation (the same described above) and subgradient optimization. The dual bound obtained is strengthened by a procedure H2, that uses the linear relaxation of the CPMP reformulation as a set partitioning problem, whose columns are the feasible clusters whose reduced costs do not exceed the gap found in H1; the reduced costs are evaluated using the best set of multipliers encountered. In order to reduce the problem size only the best Δ clusters ($\Delta = 2000$ in the implementation of [6]) for every candidate median are considered. When a new dual bound (and the corresponding dual solution) is obtained, a new set of columns is computed and an integer solution is obtained using CPLEX (procedure EHP); that solution can be non-optimal, but a dual bound based on the reduced cost of the best cluster discarded can either prove optimality or provide information on the approximation obtained.

BHMM may behave as an optimization algorithm or as a heuristic depending on the value of parameter Δ, which is user-controlled. Unfortunately it is not known how Δ must be chosen in order to have an a priori optimality guarantee: once a value of Δ has been guessed, it is possible that BHMM terminate providing a suboptimal solution together with a lower bound. If the user wants to find the optimal solution, he must guess a larger value of Δ and try again and the procedure must be repeated until optimality is reached or the available computing resources are exhausted. Obviously larger values of Δ imply larger amounts of computing time. On the contrary our branch-and-price algorithm does certainly yield the optimal solution, provided that the program is not aborted for insufficient computing resources. Hence when a difficult CPMP instance must be solved to optimality one has two alternatives: either tentatively tuning Δ according to the available resources and running BHMM in the hope that the solution will be provably optimal or running the branch-and-price algorithm in the hope that the computing resources will be sufficient. Even if the two methods are of different nature from a theoretical viewpoint, we tried to make a significant experimental comparison since at the best of our knowledge they represent the state of the art to solve the CPMP. To this purpose we set Δ to the same value indicated in [6], that is $\Delta = 2000$, which is large enough to often obtain provably optimal solutions and small enough to make the algorithms comparable from the viewpoint of computing time.

Moreover we had to address two other points, concerning the formulation of the problem and the initialization of BHMM. In [6] the authors used a slightly less general formulation of the CPMP, imposing that a vertex hosting a selected median must belong to its cluster, that is x_{jj} variables were used instead of y_j as location variables. ¿From a modelling viewpoint this assumption is a significant restriction in capacitated problems: it can change the optimal solution (this is the case of instance ccpx-7, class δ for example) and it can even inhibit the existence of feasible solutions. For this reason we decided to address the more general formulation, in which y_j variables are introduced. However from an algorithmic viewpoint the difference is negligible: we could easily adapt BHMM to the more general formulation and we did not observe any significant change in its performances.

The second key issue to make a significant experimental comparison is initialization. The computational results reported in [6] were obtained by initializing procedure H1 with very tight primal bounds: the authors used solutions given by a so-called "bionomic" algorithm described in [70] and these solutions are often optimal; the initialization was made by taking such values increased by 1 as initial upper bounds. However the computing time spent in the computation of so good initial solutions took 10 minutes on an Intel Pentium 166MHz PC, as reported in [70], and it was not considered in the presentation of the computational results in [6]. On the contrary our branch-and-price algorithm does not require the knowledge of quasi-optimal solutions in advance, because upper bounds are generated inside the column generation procedure.

Therefore to make a fair comparison between the two techniques, we did the following. First of all we initialized BHMM as in [6] and we compared the outcome with the results reported by Baldacci et al., to validate our reimplementation of their code, which had not been made available to us. The observed computing times well correspond to those reported in [6] when they are scaled by a factor 10.8, which is the ratio between the speed of our hardware and that of the machine cited in [6], according to the LINPACK benchmark [31].

Then we compared the BHMM algorithm and the branch-and-price algorithm under the usual hypothesis that nothing is known in advance: this means that the branch-and-price algorithm started from scratch with no initialization, while the BHMM algorithm was initialized with the best solution given by the bionomic algorithm (the values were taken from [6], without increasing them by 1) and its computing time was increased by a corresponding amount, equal to 10 minutes divided by the scale factor given by the LINPACK benchmark: for the machine used for the experiments reported in [70] such factor is about 15. The comparison is reported in table 6.9 and it was necessarily limited to the instances of class α for which the initial value given by the bionomic algorithm was known. As in [6], computation was halted after one hour for the tests on instances with $N = 50$ and $N = 100$, and after two hours on instances with $N = 150$ and $N = 200$. As in the previous tables, the column "v^*(gap)" reports the optimal value if optimality was proven, otherwise the best primal and dual bounds are shown and the "time" column is marked with a dash. For each instance, in the columns "Bionomic bound" and "Optimum" the primal bound given by the bionomic algorithm and the optimal value (if known) are indicated. When the BHMM algorithm terminated without proving optimality, we report both the best primal and dual bounds and the time required. CPLEX 6.5 was used both as an LP solver and a IP solver in the BHMM algorithm.

For instances with $N = 50$ both algorithms performed well: BHMM was slower, due to the time needed by the bionomic algorithm to reach a good primal bound. Branch-and-price solved to optimality one instance more than BHMM. For instances with $N = 100$ and $N = 150$ BHMM was faster, but branch-and-price proved the optimality of more instances and produced smaller primal-dual gaps. For instances with $N = 200$ both BHMM and branch-and-price solved to optimality only 3 instances over 10. Once again BHMM was faster, but branch-and-price gave tighter approximations.

6.4.4 Concluding remarks

The computational results presented in sections 6.3 and 6.4 show that the performances of all algorithms we have considered are strongly affected both by the size of the instance, that is the number of vertices, and by the value of the $\frac{p}{N}$ ratio. In addition the algorithms show complementary behaviors. CPLEX is particularly effective in solving the smaller instances and those of class β. However larger problems are still too big to be efficiently managed by a general purpose solver: no feasible solution was found for several of the instances with $N = 200$ after one hour of computation.

The Lagrangean-based branch-and-bound works fine for small values of $\frac{p}{N}$ (class α), but its performance worsens quickly as this ratio increases. In fact, both the quality of the lower bound and the effectiveness of the branching policy are affected by high $\frac{p}{N}$ ratios. The branch-and-price algorithm shows a much more stable behavior: it is effective on small problems, where no significant difference can be observed within the four classes of instances, still giving tight upper and lower bounds for the larger instances. A feasible solution is always found within a few percentage points from optimality.

This suggests to use branch-and-price as an approximation method for large CPMP problems: as an alternative approach, we investigated the behavior of the BHMM algorithm initialized with upper bounds given by the bionomic meta-heuristic. When both methods complete the computation within the time limit, the BHMM algorithm is on average faster. However branch-and-price consistently yields smaller gaps between upper and lower bounds, and is able to prove the optimality of a larger number of instances.

Instance		Class α			Class β			Class γ			Class δ		
		v^* (gap)	time (s)	status	v^* (gap)	time (s)	status	v^* (gap)	time (s)	status	v^* (gap)	time (s)	status
N = 50	cpmp01	713	4.98		383	7.26		298	2.07		266	3.96	
	cpmp02	740	0.48		412	4.34		336	16.46		298	7.31	
	cpmp03	751	31.15		405	27.41		314	8.98		311	20.27	
	cpmp04	651	1.43		384	453.59		303	98.69		277	8.43	
	cpmp05	664	5.92		429	85.84		351	14.52		356	22.81	
	cpmp06	778	0.68		482	70.51		390	23.66		370	6.39	
	cpmp07	787	50.02		445	380.05		361	45.99		358	19.69	
	cpmp08	820	154.10		403	131.38		353	571.27		312	230.11	
	cpmp09	715	26.31		436	290.84		373	174.40		412	804.70	
	cpmp10	829	127.49		461	551.10		390	530.10		458	646.66	
		0.000%	40.256	10	0.000%	200.232	10	0.000%	148.614	10	0.000%	177.033	10
N = 100	cpmp11	1006	423.55		544	969.41		414	80.13		415	696.14	
	cpmp12	966	142.38		504	2045.81		391	1691.34		377	1092.72	
	cpmp13	1026	50.01		555	1788.78		446	100.56		412	365.01	
	cpmp14	982	929.99		544	2279.67		447	1897.32		421	1624.19	
	cpmp15	1091	1111.13		583	700.60		474	323.27		496	1158.77	
	cpmp16	954	176.70		(550;529)	–	(a)	(455;447)	–	(a)	428	1261.00	
	cpmp17	1034	577.67		542	261.86		431	46.31		440	2612.34	
	cpmp18	1043	1129.22		508	223.17		456	3223.04		450	690.38	
	cpmp19	1031	557.86		551	3538.42		445	2107.70		450	1607.81	
	cpmp20	(1061;995)	–	(a)	(611;533)	–	(a)	460	1925.53		(–;458)	–	(a)
		0.663%	869.952	9	1.860%	1900.980	8	0.179%	1499.630	9	(0.000%)	1234.262	9
N = 150	cpmp21	1288	481.72		(1292;672)	–	(a)	(760;581)	–	(a)	(1466;539)	–	(a)
	cpmp22	(1256;1251)	–	(a)	(876;645)	–	(a)	(1364;535)	–	(a)	(–;575)	–	(a)
	cpmp23	1279	665.76		(706;653)	–	(a)	(1652;542)	–	(a)	(1474;535)	–	(a)
	cpmp24	(1254;1213)	–	(a)	594	1235.17		(1241;492)	–	(a)	(491;484)	–	(a)
	cpmp25	1193	430.84		629	204.04		488	1330.24		(1044;427)	–	(a)
	cpmp26	1264	1502.08		653	886.45		(750;528)	–	(a)	512	589.29	
	cpmp27	1323	3463.23		(1540;701)	–	(a)	(1967;560)	–	(a)	(–;668)	–	(a)
	cpmp28	1233	455.58		644	473.86		503	138.68		471	1199.79	
	cpmp29	1219	202.37		649	390.83		(545;544)	–	(a)	494	347.70	
	cpmp30	1201	156.56		630	896.90		(1234;487)	–	(a)	444	163.33	
		0.378%	1462.320	8	25.588%	1861.870	6	98.966%	3052.720	2	(61.680%)	2102.900	4
N = 200	cpmp31	(1439;1371)	–	(a)	(1545;710)	–	(a)	575	396.79		(726;526)	–	(a)
	cpmp32	(1472;1411)	–	(a)	(–;776)	–	(a)	(-;680)	–	(a)	(–;722)	–	(a)
	cpmp33	(1406;1363)	–	(a)	(781;701)	–	(a)	(1492;594)	–	(a)	(–;545)	–	(a)
	cpmp34	(2466;1372)	–	(a)	(–;762)	–	(a)	(832;666)	–	(a)	(–;776)	–	(a)
	cpmp35	(1494;1431)	–	(a)	(1462;723)	–	(a)	(–;585)	–	(a)	(1781;533)	–	(a)
	cpmp36	1382	1089.73		701	2949.48		(662;573)	–	(a)	(–;534)	–	(a)
	cpmp37	1458	936.61		753	1809.92		(1341;614)	–	(a)	(1464;578)	–	(a)
	cpmp38	(–;1370)	–	(a)	(1533;729)	–	(a)	(646;606)	–	(a)	(–;576)	–	(a)
	cpmp39	1374	976.39		(835;712)	–	(a)	(610;584)	–	(a)	(563;536)	–	(a)
	cpmp40	1416	2878.57		(1484;736)	–	(a)	(678;614)	–	(a)	(–;563)	–	(a)
		(10.731%)	2707.790	4	(57.554%)	3368.600	2	(41.440%)	3285.100	1	(107.613%)	3697.240	0

(a) = computation exceeded time limit; (b) = out of memory

Table 6.7: CPLEX 6.5

Instance		class α			class β			class γ			class δ		
		v^*(gap)	time (s)	status	v^*(gap)	time (s)	status	v^*(gap)	time (s)	status	v^*(gap)	time (s)	status
N = 50	cpmp01	713	0.17		383	12.35		298	3.4		266	440.37	
	cpmp02	740	0.06		412	1.19		336	271.86		298	2423.56	
	cpmp03	751	0.13		405	8.85		314	18.49		311	77.54	
	cpmp04	651	0.07		384	910.32		303	85.28		277	256.79	
	cpmp05	664	0.09		429	308.33		351	22.15		356	216.23	
	cpmp06	778	0.09		482	256.78		390	272.26		370	39.49	
	cpmp07	787	0.8		445	715.14		361	107.85		358	42.29	
	cpmp08	820	17.54		403	153.58		(365;322)	-	(a)	(313;290)	-	(a)
	cpmp09	715	0.92		436	91.26		373	165.14		(418;365)	-	(a)
	cpmp10	829	2.85		461	647.89		(390;368)	-	(a)	(481;403)	-	(a)
		0.000%	2.272	10	0.000%	310.569	10	1.933%	118.304	8	4.181%	499.467	7
N = 100	cpmp11	1006	3.26		(544;527)	-	(a)	(439;405)	-	(a)	(435;400)	-	(a)
	cpmp12	966	15.26		(509;496)	-	(a)	(394;374)	-	(a)	(393;362)	-	(a)
	cpmp13	1026	1.32		(567;534)	-	(a)	(451;439)	-	(a)	(425;402)	-	(a)
	cpmp14	982	47.97		(548;529)	-	(a)	(481;433)	-	(a)	(442;411)	-	(a)
	cpmp15	1091	32.33		(593;570)	-	(a)	(481;468)	-	(a)	(504;482)	-	(a)
	cpmp16	954	8.17		(558;518)	-	(a)	(453;430)	-	(a)	(430;423)	-	(a)
	cpmp17	1034	42.49		542	1673.37		(439;424)	-	(a)	(795;427)	-	(a)
	cpmp18	1043	39.94		508	977.96		(493;429)	-	(a)	(500;429)	-	(a)
	cpmp19	1031	18.92		(562;531)	-	(a)	(473;430)	-	(a)	(498;437)	-	(a)
	cpmp20	1005	3377.63		(597;514)	-	(a)	(483;448)	-	(a)	(1630;463)	-	(a)
		0.000%	358.729	10	4.936%	1325.660	2	7.196%	-	0	40.554%	-	0
N = 150	cpmp21	1288	227.11		(683;674)	-	(a)	(626;580)	-	(a)	(585;543)	-	(a)
	cpmp22	1256	1100.95		(665;643)	-	(a)	(599;537)	-	(a)	(992;582)	-	(a)
	cpmp23	1279	62.43		(711;639)	-	(a)	(659;548)	-	(a)	(1235;528)	-	(a)
	cpmp24	1220	288.73		(603;585)	-	(a)	(533;495)	-	(a)	(518;471)	-	(a)
	cpmp25	1193	29.55		(634;627)	-	(a)	(499;484)	-	(a)	(461;427)	-	(a)
	cpmp26	1264	48.11		(653;646)	-	(a)	(565;526)	-	(a)	(530;504)	-	(a)
	cpmp27	1323	1795.07		(837;711)	-	(a)	(631;566)	-	(a)	(2859;694)	-	(a)
	cpmp28	1233	74.16		(647;630)	-	(a)	(526;498)	-	(a)	(493;462)	-	(a)
	cpmp29	1219	11.70		(656;641)	-	(a)	(565;529)	-	(a)	(505;488)	-	(a)
	cpmp30	1201	11.61		(641;619)	-	(a)	(515;487)	-	(a)	(472;434)	-	(a)
		0.000%	364.942	10	4.762%	-	0	8.758%	-	0	56.609%	-	0
N = 200	cpmp31	1378	482.48		(767;711)	-	(b)	(590;570)	-	(a)	(590;527)	-	(b)
	cpmp32	(1447;1404)	-	(a)	(1580;789)	-	(b)	(1699;694)	-	(b)	(- ;732)	-	(b)
	cpmp33	(1385;1360)	-	(a)	(765;693)	-	(b)	(751;598)	-	(a)	(662;547)	-	(a)
	cpmp34	(1385;1372)	-	(a)	(1137;770)	-	(b)	(915;677)	-	(a)	(- ;788)	-	(b)
	cpmp35	1437	2982.44		(801;725)	-	(b)	(636;592)	-	(a)	(630;540)	-	(b)
	cpmp36	1382	100.30		(717;690)	-	(b)	(615;569)	-	(a)	(574;537)	-	(a)
	cpmp37	1458	259.51		(783;743)	-	(a)	(679;615)	-	(b)	(668;578)	-	(b)
	cpmp38	(1390;1369)	-	(a)	(839;730)	-	(b)	(645;595)	-	(b)	(764;581)	-	(b)
	cpmp39	1374	106.60		(736;712)	-	(a)	(620;573)	-	(b)	(600;528)	-	(b)
	cpmp40	1416	1300.01		(842;735)	-	(b)	(712;606)	-	(b)	(1366;568)	-	(b)
		0.738%	871.89	6	21.882%	-	0	26.908%	-	0	(32.217%)	-	0

(a) = computation exceeded time limit; (b) = out of memory

Table 6.8: Branch-and-bound with Lagrangean relaxation

Problem					BHMM + bionomic bound		Branch-and-price	
N	p	Instance	Bionomic bound	Optimum	v^* (gap)	time(s)	v^* (gap)	time(s)
50	5	ccpx1	713	713	713	40.25	713	4.58
		ccpx2	740	740	720	40.08	740	0.16
		ccpx3	751	751	751	40.16	751	2.43
		ccpx4	651	651	651	40.07	651	0.31
		ccpx5	664	664	664	40.15	664	0.56
		ccpx6	778	778	778	40.08	778	0.33
		ccpx7	787	787	787	41.24	787	7.13
		ccpx8	820	820	(820;790)	-	820	1904.73
		ccpx9	715	715	715	40.28	715	1.00
		ccpx10	829	829	829	44.41	829	7.96
		Average			0.38%	40.74	0.00%	2.72
100	10	ccpx11	1006	1006	1006	42.20	1006	19.08
		ccpx12	966	966	966	133.83	966	93.51
		ccpx13	1026	1026	1026	40.62	1026	30.46
		ccpx14	982	982	982	125.23	982	232.13
		ccpx15	1091	1091	1091	90.04	1091	407.73
		ccpx16	954	954	954	40.95	954	5.70
		ccpx17	1034	1034	1034	57.67	1034	113.91
		ccpx18	1043	1043	(1043;1040)	179.14	1043	877.95
		ccpx19	1031	1031	1031	43.48	1031	18.03
		ccpx20	1013	1005	(1013;985)	-	(1006;993)	-
		Average			0.31%	71.75	0.13%	115.07
150	15	ccpx21	1290	1283	(1283;1279)	3952.79	1283	4119.65
		ccpx22	1292	1291	1291	206.04	1291	450.46
		ccpx23	1220		(1219;1192)	-	(1216;1207)	-
		ccpx24	1236	1235	(1235;1230)	470.59	1235	5143.87
		ccpx25	1189	1188	1188	40.72	1188	3.94
		ccpx26	1228	1227	1227	40.63	1227	4.37
		ccpx27	1270	1269	1269	208.40	1269	730.71
		ccpx28	1181	1180	1180	49.49	1180	391.48
		ccpx29	1260		(1259;1248)	6389.09	(1262;1250)	-
		ccpx30	1243	1241	(1241;1236)	2291.95	(1241;1239)	-
		Average			0.57%	109.05	0.19%	316.19
200	20	ccpx31	1447	1446	1446	294.68	1446	3827.76
		ccpx32	1352		(1351;1336)	-	(1358;1344)	-
		ccpx33	1391	1390	1390	40.45	1390	10.47
		ccpx34	1395		(1394;1362)	-	(1393;1374)	-
		ccpx35	1401		(1400;1382)	-	(1401;1389)	-
		ccpx36	1384	1382	(1382;1378)	2643.33	(1382;1380)	-
		ccpx37	1399		(1398;1370)	-	(1388;1379)	-
		ccpx38	1462		(1461;1438)	-	(1476;1447)	-
		ccpx39	1427		(1426;1421)	1013.62	(1426;1425)	-
		ccpx40	1393	1392	1392	41.20	1392	12.94
		Average			0.91%	125.44	0.62%	1283.72

Table 6.9: Comparison between BHMM and branch-and-price

Appendix

Primal heuristic:

```
Input:  $z^k \ \forall k \in Z$
(the solution of LRMP)
Output:  $C^j \ \forall j \in \mathcal{M}$
(the set of clusters of a feasible solution for CPMP)

(Step 1: medians selection)
$f_{ij} = \sum_{k \in Z^j} x_i^k z_k^j \quad \forall i \in \mathcal{N} \ \forall j \in \mathcal{M}$
$\psi_j = \sum_{i \in \mathcal{N}} f_{ij} \quad \forall j \in \mathcal{M}$
$\bar{\mathcal{M}}$ := $\emptyset$
for $p$ times do
    $j^*$ := argmax$_{j \in \mathcal{M} \setminus \bar{\mathcal{M}}}\{\psi_j\}$
    $\bar{\mathcal{M}}$ := $\bar{\mathcal{M}} \cup \{j^*\}$

(Step 2: direct assignment)
forall $j \in \bar{\mathcal{M}}$ do $q_j := Q_j$
$\mathcal{N}_W := \mathcal{N}$
$C^1$ := $C^2$ := ... := $C^M$ := $\emptyset$
while $\mathcal{N}_W \neq \emptyset$ do
      (evaluation of regret values)
      forall $i \in \mathcal{N}_W$ do
              $\mathcal{M}^i = \{j \mid j \in \bar{\mathcal{M}}, q_j \geq w_i\}$
              if $\mathcal{M}^i = \emptyset$ then GOTO step 3
              $j_i'$ := argmax$_{j \in \mathcal{M}^i}\{f_{ij}\}$
              if $\mathcal{M}^i = \{j_i'\}$ then $D_i$ := $+\infty$
              else
                    $j_i''$ := argmax$_{j \in \mathcal{M}^i, j \neq j_i'}\{f_{ij}\}$
                    $D_i$ := $f_{ij'} - f_{ij''}$
      (assignment of the node with highest regret)
      $i^*$ := argmax$_{i \in \mathcal{N}_W}\{D_i\}$
      $j^*$ := $j_{i^*}'$
      $C^{j^*}$ := $C^{j^*} \cup \{i^*\}$
      $\mathcal{N}_W$ := $\mathcal{N}_W \setminus \{i^*\}$
      $q_{j^*}$ := $q_{j^*} - w_{i^*}$

(Step 3: assignment through exchanges)
while $\mathcal{N}_W \neq \emptyset$ do
      forall $i \in \mathcal{N}_W$ do
              (Evaluation of the set of clusters in which node $i$ could be inserted)
              $L_i$ = $\{j \mid \exists k \in C^j \mid q_j + w_k \geq w_i, w_k < w_i, j \in \bar{\mathcal{M}}\}$
              forall $j \in L_i$ do
                     $l(i,j)$ := min $_{k \in C^j}\{(q_j + w_k) \mid q_j + w_k \geq w_i, w_k < w_i\}$
                     $k(i,j)$ := argmin$\{l(i,j)\}$
       if $\bigcup_{i \in \mathcal{N}_W} L_i \neq \emptyset$ then
              (Shifting that minimizes the residual capacity of a median)
              $(i^*, j^*)$ := argmin$_{i \in \mathcal{N}_W, j \in L_i}\{l(i,j) - w_i\}$
              $C^{j^*}$ := $C^{j^*} \setminus \{k(i^*, j^*)\}$; $C^{j^*}$ := $C^{j^*} \cup \{i^*\}$
              $q_{j^*}$ := $l(i^*, j^*) - w_{i^*}$
              $\mathcal{N}_W$ := $\mathcal{N}_W \setminus \{i^*\}$
              if $\exists j \in \bar{\mathcal{M}} \mid q_j \geq w_{k(i^*,j^*)}$ then
                     $C^j$ := $C^j \cup \{k(i^*, j^*)\}$
                     $q_j$ := $q_j - w_{k(i^*,j^*)}$
              else $\mathcal{N}_W$ := $\mathcal{N}_W \cup \{k(i^*, j^*)\}$
       else FAIL

(Step 4: solution improvement)
if $\mathcal{N}_W = \emptyset$ then
 (definition of a dummy node $\Omega$)
 $w_\Omega = 0$; $d_{\Omega j} = 0 \ \forall j \in \mathcal{M}$

 forall $j \in \bar{\mathcal{M}}$, forall $i \in C^j$ do    $s_i$ := $j$
 do
       forall $i \in \mathcal{N}$ do
              forall $j \in \bar{\mathcal{M}}$ do
                     $E_i^j$ = $\{k \in C^j \cup \{\Omega\} \mid q_j + w_k \geq w_i, q_{s_i} + w_i \geq w_k\}$
              $g_i$ := min $_{(j \in \bar{\mathcal{M}}, k \in E_i^j)}\{d_{ij} - d_{is_i} + d_{ks_i} - d_{kj}\}$
              $(j_i', k_i')$ := argmin$_{(j \in \bar{\mathcal{M}}, k \in E_i^j)}\{d_{ij} - d_{is_i} + d_{ks_i} - d_{kj}\}$
       $i^*$ := argmin$_{i \in \mathcal{N}}\{g_i\}$; $j^*$ := $j_{i^*}'$; $k^*$ := $k_{i^*}'$
       if $(g_{i^*} < 0)$ then
              $q_{s_{i^*}}$ := $q_{s_{i^*}} + w_{i^*} - w_{k^*}$
              $q_{j^*}$ := $q_{j^*} - w_{i^*} + w_{k^*}$
              if $(k^* \neq \Omega)$ then
                            $C^{j^*}$ := $C^{j^*} \setminus \{k^*\}$; $C^{s_{i^*}}$ := $C^{s_{i^*}} \cup \{k^*\}$
                            $s_{k^*}$ := $s_{i^*}$
              $C^{s_{i^*}}$ := $C^{s_i} \setminus \{i^*\}$; $C^{j^*}$ := $C^{j^*} \cup \{i^*\}$;
              $s_{i^*}$ := $j^*$
 while $g_{i^*} < 0$
```

Chapter 7

A computational evaluation of a general branch-and-price framework for capacitated network location problems

The purpose of this paper is to illustrate a general framework for network location problems, based on column generation and branch-and-price. In particular we consider capacitated network location problems with single-source constraints. We consider several different network location models, by combining cardinality constraints, fixed set-up costs, concentrator restrictions and regional constraints. Our general branch-and-price-based approach can be seen as a natural counterpart of the branch-and-cut-based commercial ILP solvers, with the advantage of exploiting the tightness of the lower bound provided by the set partitioning reformulation of network location problems. Branch-and-price and branch-and-cut are compared through an extensive set of experimental tests.

7.1 Introduction

Finding the optimal location for facilities like warehouses or servers in distribution or telecommunication networks and deciding how to allocate clients to them is a very complex task, since it requires to solve $\mathcal{NP}$-hard combinatorial problems. Classical and well-studied examples are the Capacitated Facility Location Problem and the P-Median Problem [59]; integer linear programming has shown to be the most appropriate tool for modelling these problems. When tackling an integer linear programming problem, one has the choice between using a general-purpose solver and designing an "ad hoc" algorithm. The most common general-purpose solvers for integer linear programming problems, like ILOG CPLEX and others, use branch-and-cut methods. They combine powerful linear programming solvers with suitable subroutines able to detect violated inequalities and branching strategies to develop a search tree.

The purpose of this paper is to illustrate a general framework for network location problems, which is based on column generation and branch-and-price. In particular we concentrate on capacitated network location problems with single-source constraints. We consider several different network location models, by combining cardinality constraints, fixed set-up costs, concentrator restrictions and regional constraints and we illustrate how all of them can be solved in almost the same way by a generalization of a branch-and-price algorithm we recently designed for the Capacitated P-Median Problem [18].

It is clear that for each particular problem we consider it should be possible to develop more effective "ad hoc" algorithms, both following the branch-and-cut approach and the branch-and-price one (and possibly others, such as Lagrangean relaxation and branch-and-bound): however the accent here is put on generality and flexibility. Our goal in this study was to develop a kind of branch-and-price equivalent of the branch-and-cut-based commercial ILP solvers. Apart from being available for free for research purposes, our branch-and-price framework has the advantage of exploiting the tightness of the lower bound provided by the set partitioning reformulation of network location problems.

The outline of the paper is as follows. In Section 7.2 we report the models of the single-source capacitated location problems we consider and we review the relevant literature and the solution methods proposed so far. In Section 3 we describe our branch-and-price framework. In Section 4 we review the main algorithmic features of a general-purpose ILP solver we used as a benchmark. In Section 5 we present our experimental results. Conclusions are outlined in Section 6.

7.2 Single-source capacitated location problems

In a basic facility location scenario the best trade-off has to be found between the cost for building facilities at certain sites and the cost to serve customers. Here we consider min-sum objective functions, where service costs are assumed to be proportional to the distance between each customer and the facility to which it is assigned. The cost of building facilities can be taken into account into two different ways: either with an additional term in the objective function or with a constraint (or both). In the former case fixed set-up costs are specified for each candidate site; these costs are to be payed when a facility is set up at that site. In the latter case a cardinality constraint is added to the model, so that the number of available facilities is bounded from above. In this section we briefly review the mathematical formulations of network facility location problems with fixed set-up costs and cardinality constraints. Then we also outline the formulation of the same problems when regional constraints are added. All problems we consider have capacitated facilities and single-source constraints, so that it is not allowed to split customers' demands on more than one facility.

The following definitions apply to all models considered in the remainder of this paper. We are given a set $\mathcal{N}$ of customers and a set $\mathcal{M}$ of candidate sites where facilities can be located. An integer weight w_i represents the demand of each customer $i \in \mathcal{N}$. The capacity of a facility built in each site $j \in \mathcal{M}$ is represented by an integer Q_j. Integer coefficients d_{ij} (usually referred to as *distances*) describe the cost of allocating customer $i \in \mathcal{N}$ to a facility located in site $j \in \mathcal{M}$. We assume that $d_{ij} \geq 0 \; \forall i \in \mathcal{N}, j \in \mathcal{M}$.

7.2.1 Fixed set-up costs

The set-up cost for a facility in each site $j \in \mathcal{M}$ is represented by an integer coefficient f_j. The problem in which fixed set-up costs are incurred is known as Capacitated Facility Location Problem with Single Source constraints (SS-CFLP).

A formulation for the SS-CFLP is the following:

$$
\begin{aligned}
SS-CFLP) \quad \min \quad & v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} + \sum_{j \in \mathcal{M}} f_j y_j && && (7.1) \\
\text{s.t.} \quad & \sum_{j \in \mathcal{M}} x_{ij} = 1 && \forall i \in \mathcal{N} && (7.2) \\
& \sum_{i \in \mathcal{N}} w_i x_{ij} \leq Q_j y_j && \forall j \in \mathcal{M} && (7.3) \\
& x_{ij} \in \{0, 1\} && \forall i \in \mathcal{N}, \forall j \in \mathcal{M} && (7.4) \\
& y_j \in \{0, 1\} && \forall j \in \mathcal{M} && (7.5)
\end{aligned}
$$

Binary variables x are assignment variables: $x_{ij} = 1$ if and only if customer i is served by a facility located in site j. Binary variables y correspond to location decisions: $y_j = 1$ if and only if site j is selected to host a facility. The objective is to minimize the sum of allocation costs depending on x variables and set-up costs depending on y variables. Set partitioning constraints (7.2) impose that each customer is assigned to a facility. Capacity constraints (7.3) impose that the sum of the customers' demands assigned to a same facility does not exceed the capacity of the facility; these constraints also forbid the assignment of customers to sites which do not host facilities.

To strengthen the linear relaxation of this formulation, disaggregated inequalities

$$x_{ij} \leq y_j, \ \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{7.6}$$

are introduced. They arise from constraints (7.3), and the aggregated demand constraint

$$\sum_{j \in \mathcal{M}} Q_j y_j \geq \sum_{i \in \mathcal{N}} w_i. \tag{7.7}$$

When single-source constraints are relaxed, that is x variables have a continuous domain in the range $[0, 1]$, the Capacitated Facility Location Problem (CFLP) arises. The CFLP has been extensively studied and the literature on it is quite rich. Erlenkotter [93] proposed an algorithm in which the continuous relaxation of (7.1) - (7.6) is used as a dual bound. Effective optimization algorithms for the CFLP have been developed through Lagrangean methods [22] and cross decomposition [94]. These algorithms allow to solve problem instances involving up to 50 customers and 50 candidate sites. More recent approaches are due to Aardal [2], who exploits the polyhedral structure of the problem to design an effective branch-and-cut algorithm, and to Klose et al. [60], who apply Dantzig-Wolfe decomposition and column generation to obtain better bounds. Aardal [2] attacked instances with up to 100 customers and 75 facilities, proving the optimality of the solutions provided; Klose et al. [60] obtained good approximations on problems with up to 500 customers and 200 candidate facilities, and solved to optimality problem instances with up to 200 customers embedding their column generation routine in a branch-and-price algorithm. Daskin [25] and Drexl [59] give detailed surveys on the CFLP.

The methods proposed in the literature for the single-source constraints could solve only smaller instances. Also in this case, Lagrangean relaxation can be used to design branch-and-bound methods. Dualization of the capacity constraints (7.3) is discussed in Klincewitz et al. [58], where the authors solve problem instances with 50 customers in few minutes. Pirkul [86] obtained better results by dualizing the partitioning constraints (7.2): he could solve instances with 100 customers and 20 candidate facilities. Holmberg et al. [49] proposed to couple a Lagrangean relaxation of the capacity constraints with a repeated matching heuristic, to solve problem instances involving up to 200 customers and 30 candidate facilities in some minutes. A column generation approach for the SS-CFLP was proposed by Neebe and Rao [82], who solved problem instances with more than 35 customers and 25 facilities. More recently Fernandez and Diaz [28] implemented a new branch-and-price method, which solved to optimality instances with up to 90 customers and 30 facilities in a few hours.

7.2.2 Concentrators

When the set of customers and the set of candidate sites coincide, the resulting problem is called Capacitated Concentrator Location Problem (CCLP) [45]. This variant has received much attention in telecommunication networks design, where facilities represent electronic devices and customers represent terminals on a telecommunication network. In the CCLP each location variable y_j can be replaced by a corresponding assignment variable x_{jj}. Although some authors [86] used the name "Capacitated Concentrators" to indicate the SS-CFLP, we follow [63] and [62] and we indicate as "concentrator problems" the models in which variables x_{jj} replace variables y_j to represent location decisions.

The mathematical formulation of the CCLP is as follows:

$$CCLP) \quad \min \quad v = \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{M}} d_{ij}x_{ij} + \sum_{j\in\mathcal{M}} f_j x_{jj}$$

$$\text{s.t. } (7.2), (7.4)$$

$$\sum_{i\in\mathcal{N}} w_i x_{ij} \le Q_j x_{jj} \qquad \forall j \in \mathcal{M} \tag{7.8}$$

$$x_{ij} \le x_{jj} \qquad \forall i \in \mathcal{N}, j \in \mathcal{M} \tag{7.9}$$

$$\sum_{j\in\mathcal{M}} Q_j x_{jj} \ge \sum_{i\in\mathcal{N}} w_i \tag{7.10}$$

The polyhedral structure of these problems has recently been studied in detail by Labbé and Yaman [63] [62]. Problems on networks with up to 100 terminals can be solved to optimality with a branch-and-cut approach in half an hour of CPU time.

7.2.3 Cardinality constraints

When the set-up costs are taken into account through a limit on the number of facilities that can be built, the resulting problems are usually called "median problems". The most basic problem of this type is the p-Median Problem (PMP), which is the discrete counterpart of the famous multi-source Fermat-Weber problem [90]. The PMP consists in partitioning the vertices of a given graph into p subsets and to choose a *median* vertex in each subset, minimizing the sum of the distances between each vertex of the graph and the median of its subset. The PMP arises in many different contexts such as network design, telecommunications, distributed database design, transportation and distribution logistics. Kariv and Hakimi [53] proved that the PMP is $\mathcal{NP}$-hard. Optimization algorithms based on Lagrangean relaxation have been proposed by Narula et al. [80], Cornuéjols et al. [24], Christofides and Beasley [21] and Beasley [9]; approaches based on dual formulations are illustrated in Galvao [40] and Hanjoul and Peeters [46]. Among the most recent contributions we cite the branch-and-cut-and-price algorithm by Avella et al. [3], the branch-and-price algorithm by Senne et al. [97] and the semi-Lagrangean relaxation algorithm by Beltran et al. [10]. The most successful approach is that of Avella et al.: an instance on a graph with 3795 vertices, in which 150 facilities have to be selected was solved, even though this optimization took several hours of CPU time. A survey on the PMP can be found in Labbé et al. [61].

Here we consider the capacitated version of the problem, that is the Capacitated PMP (CPMP), that can be formulated as follows.

$$
\begin{aligned}
CPMP) \quad \min \quad & v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} \\
\text{s.t.} \quad & (7.2), (7.3), (7.4), (7.5), (7.6) \\
& \sum_{j \in \mathcal{M}} y_j = p
\end{aligned}
$$

Algorithms devised for the uncapacitated PMP cannot be adapted to the CPMP in a straightforward way: even finding a feasible solution is $\mathcal{NP}$-complete when capacities are considered. Recent contributions to the literature on the CPMP include the algorithm of Baldacci et al. [6], which can give "a posteriori" guarantee on optimality or the approximation achieved and a branch-and-price algorithm of Ceselli and Righini [18]. The branch-and-price approach solved problems on graphs with up to 200 vertices and any number of medians in less than two hours of computation.

Very recently, two heuristic algorithms for the CPMP have been presented: Lorena and Senne [69] followed a column generation approach, finding good solutions on real instances with up to 402 vertices, while Diaz and Fernández [29] attacked an instance with 737 vertices through hybrid scatter search and path relinking.

Both Baldacci et al. [6] and Lorena and Senne [69] replaced the location variables y_j with the assignment variables x_{jj}, so that each median is forced to be coincident with one of the vertices of its cluster. To be consistent with the terminology explained above, we indicate this problem as the Capacitated P-Concentrator Location Problem (CPCLP).

$$
\begin{aligned}
CPCLP) \quad \min \quad & v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} \\
\text{s.t.} \quad & (7.2), (7.8), (7.4), (7.9), (7.10) \\
& \sum_{j \in \mathcal{M}} x_{jj} = p
\end{aligned}
$$

As reported by Ceselli [15], this restriction can change the optimal solution or even inhibit the existence of feasible solutions.

7.2.4 Regional Constraints

A region is defined as a subset of candidate sites; a regional constraint imposes an upper or lower bound to the number of facilities that can be built in a certain region. Regional constraints are usually employed to enforce equity in the geographical distribution of the facilities. Following Syam [99] and Murray and Gerrard [79], we formulate the SS-CFLP with regional constraints (RCSS-CFLP) as follows:

$$
\begin{aligned}
RCSS - CFLP) \quad \min \quad & v = \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{M}} d_{ij} x_{ij} + \sum_{j \in \mathcal{M}} f_j y_j \\
\text{s.t.} \quad & (7.2), (7.3), (7.4), (7.5), (7.6), (7.7) \\
& \sum_{j \in R} y_j \le u_R \qquad \forall R \in \mathcal{R} \qquad (7.11) \\
& \sum_{j \in R} y_j \ge l_R \qquad \forall R \in \mathcal{R} \qquad (7.12)
\end{aligned}
$$

Each $R \in \mathcal{R}$ represents a region, that is a subset of the candidate sites. Constraints (7.11) and (7.12) impose respectively an upper and a lower bound on the number of facilities that can be located in each region. It is worth noting that in general regions may overlap, while the methods proposed in Syam [99] and Murray and Gerrard [79] are restricted to the case of non-overlapping regions.

Regional constraints can be also added to all the other location problems listed above. The CPMP itself can be viewed as a RCSS-CFLP in which there are no set-up costs ($f_j = 0 \ \ \forall j \in \mathcal{M}$) and just one region R with $u_R = l_R = p$.

7.2.5 Set partitioning formulation

All location problems described in the previous section admit an alternative formulation set partitioning formulation, in which each column corresponds to a feasible cluster, that is an assignment of customers to a facility, that satisfies the capacity constraint. A binary variable z_k^j is associated with each cluster. Let us indicate by Z^j the set of all feasible clusters whose correspondent facility is located in site $j \in \mathcal{M}$. Each cluster k is described by assignment coefficients x_i^k equal to 1 if and only if customer $i \in \mathcal{N}$ belongs to cluster k. The set partitioning formulation is the following.

$$
\begin{aligned}
MP) \ \min \quad & \sum_{j \in \mathcal{M}} \sum_{k \in Z^j} (f_j + \sum_{i \in \mathcal{N}} d_{ij} x_i^k) z_k^j && && (7.13) \\
\text{s.t.} \quad & \sum_{j \in \mathcal{M}} \sum_{k \in Z^j} x_i^k z_k^j = 1 && \forall i \in \mathcal{N} && (7.14) \\
& -\sum_{k \in Z^j} z_k^j \geq -1 && \forall j \in \mathcal{M} && (7.15) \\
& -\sum_{j \in R} \sum_{k \in Z^j} z_k^j \geq -u_R && \forall R \in \mathcal{R} && (7.16) \\
& \sum_{j \in R} \sum_{k \in Z^j} z_k^j \geq l_R && \forall R \in \mathcal{R} && (7.17) \\
& z_k^j \in \{0, 1\} && \forall j \in \mathcal{M}, \ \forall k \in Z^j. && (7.18)
\end{aligned}
$$

Constraints (7.14) guarantee that each customer is assigned to a facility; constraints (7.15) impose that no more than one cluster is associated to the same facility site. For each region $R \in \mathcal{R}$, constraints (7.16) and (7.17) impose that the number of clusters with facilities in R is between a lower bound l_R and an upper bound u_R.

For concentrator-like models, each cluster always contains the corresponding facility. In this case, constraints (7.15) are redundant, because they are implied by constraints (7.14), and they can be removed. When this condition is not enforced, partitioning constraints (7.14) can be replaced by covering constraints:

$$
\sum_{j \in \mathcal{M}} \sum_{k \in Z^j} x_i^k z_k^j \geq 1 \quad \forall i \in \mathcal{N} \qquad (7.19)
$$

because all distances are non-negative and therefore it does always exist an optimal solution in which no customer is assigned more than once.

The set partitioning reformulation is the starting point for developing column generation and branch-and-price algorithms. Many authors have followed this path to develop effective algorithms to solve the SS-CFLP (Diaz and Fernandez [28]) the PMP (Senne et al. [97]) the CPMP (Baldacci et al. [6], Ceselli and Righini [18]) or similar problems (Savelsbergh [95]).

Our purpose however is not to develop an algorithm tailored to any of such problems, but rather to exploit the generality of the branch-and-price approach. Branch-and-price is applicable to all models outlined above; this suggested us to developed and evaluate a general-purpose branch-and-price solver for network location problems.

7.3 A branch-and-price algorithm

In this section we present a general branch-and-price framework for single-source capacitated location problems. This framework derives from the algorithm proposed in [18] for the exact optimization of the CPMP.

We describe the main components of the algorithm. These include the column generation subroutine, the branching strategy and the policy for the management of columns. We also discuss the use of Lagrangean lower bounds and primal heuristics. In this section we refer to set covering reformulation defined by the objective function (7.13) and by set covering constraints (7.19), convexity constraints (7.15), regional constraints (7.16) and (7.17) and integrality requirements (7.18). The cardinality constraint has not been taken into account in an explicit way, since it is a special case of regional constraints.

7.3.1 Column generation

Each set Z^j of feasible clusters served from a facility located in site j contains an exponential number of elements. Since the linear relaxation of the master problem MP (indicated hereafter by LMP) cannot be solved directly because of the exponential number of its columns, column generation is applied (see Gilmore and Gomory [43]): a restricted linear master problem (RLMP), defined by a relatively small subset of columns, is considered and solved to optimality; then, a search is performed for new columns of negative reduced cost and, if any such column is found, it is inserted into the formulation and the RLMP is solved again. When no columns of negative reduced cost exist, the optimal solution of the RLMP is also optimal for the LMP and its value is a valid lower bound to be used in a branch-and-bound framework.

The main advantage of the branch-and-price approach consists in the tightness of the lower bound. The set partitioning formulation can be obtained from the compact formulation by applying Dantzig-Wolfe decomposition [77]. As far as the linear relaxations of the two formulations are concerned, the polyhedra Ω^j described by the capacity constraints in the compact formulation are convexified in the set partitioning formulation. Since each Ω^j is the polyhedron of the linear relaxation of a binary knapsack problem (see [75] for a classical reference), which is known not to have the integrality property, its extreme points can have fractional coordinates; therefore the lower bound computed after the convexification of each set Ω^j can be tighter than that provided by the linear relaxation of the compact formulation. Many experiments [15] [18] show that this is actually the case (see also Section 7.5).

The implementation of the column generation algorithm requires the repeated solution of a pricing problem. Let $\boldsymbol{\lambda} \in \mathbb{R}_+^{|\mathcal{N}|}$, $\boldsymbol{\mu} \in \mathbb{R}_+^{|\mathcal{M}|}$, $\boldsymbol{\gamma^u} \in \mathbb{R}_+^{|\mathcal{R}|}$ and $\boldsymbol{\gamma^l} \in \mathbb{R}_+^{|\mathcal{R}|}$ be the vectors of non-negative dual variables corresponding to constraints (7.19), (7.15), (7.16) and (7.17) respectively; the reduced cost of column $k \in Z^j$ is

$$r^k(\boldsymbol{\lambda}, \boldsymbol{\mu}, \boldsymbol{\gamma^u}, \boldsymbol{\gamma^l}) = f_j + \sum_{i\in\mathcal{N}} d_{ij}x_i^k - \sum_{i\in\mathcal{N}} \lambda_i x_i^k + \mu_j - \sum_{R\in\mathcal{R}|j\in R} (\gamma_R^l - \gamma_R^u)$$

To find columns with negative reduced cost, one must solve a pricing problem for each candidate facility $j \in \mathcal{M}$:

$$\begin{aligned} \min \quad & \sum_{i\in\mathcal{N}} (d_{ij} - \lambda_i)x_i^k + f_j + \mu_j - \sum_{R\in\mathcal{R}|j\in R} (\gamma_R^l - \gamma_R^l) \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}} w_i x_i^k \le Q_j \\ & x_i^k \in \{0,1\} \qquad\qquad \forall i \in \mathcal{N} \end{aligned}$$

and this requires the solution of the following binary knapsack problem:

$$\begin{aligned} KP_j) \quad \max \quad & \tau_j = \sum_{i\in\mathcal{N}} (\lambda_i - d_{ij})x_i^k \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}} w_i x_i^k \le Q_j \\ & x_i^k \in \{0,1\} \qquad\qquad \forall i \in \mathcal{N} \end{aligned}$$

When a concentrator-like model is considered, each site selected to host a facility must belong to its cluster and this can easily be handled by fixing $x_j^k = 1$ in each KP_j.

7.3.2 Branching scheme

The optimal solution of the RLMP can be fractional; hence the column generation routine is embedded in a branch-and-bound algorithm. Different branching rules can be applied ([14] [18]). The choice of a suitable branching strategy is definitely critical because branching must not destroy the combinatorial structure of the pricing problem. The branching rule we used works as follows: for each customer

$i \in \mathcal{N}$ we consider the set $\mathcal{J}_i$ of candidate facilities for which there is a fractional assignment in the optimal solution of the RLMP:

$$\mathcal{J}_i = \{j : \sum_{k \in Z^j} z_k^j x_i^k > 0\}$$

Then we partition this set into two subsets $\mathcal{J}_i'$ and $\mathcal{J}_i''$: aiming at a balanced partition, we sort the elements of $\mathcal{J}_i$ by non-increasing value of fractional assignment and we insert them alternately in the first and in the second subset. We measure the unbalance of the partition as

$$U_i = |\sum_{j \in \mathcal{J}_i'} \sum_{k \in Z^j} z_k^j x_i^k - \sum_{j \in \mathcal{J}_i''} \sum_{k \in Z^j} z_k^j x_i^k|$$

that is the absolute difference between the value of fractional assignment to the facilities in the first subset and the facilities in the second subset. The customer i^* selected for branching is the one which produces the most balanced partition:

$$i^* \in \operatorname{argmin}_{i \in \mathcal{N}} \{U_i\}$$

Also the set of remaining candidate facilities, that is $\hat{\mathcal{J}}_{i^*} = \mathcal{M} \backslash \mathcal{J}_{i^*}$, is partitioned into two subsets $\hat{\mathcal{J}}_{i^*}'$ and $\hat{\mathcal{J}}_{i^*}''$. Then we generate two subproblems and in each of them we forbid the assignment of the branching customer i^* to the facilities of $\mathcal{J}_{i^*}' \cup \hat{\mathcal{J}}_{i^*}'$ in one branch and $\mathcal{J}_{i^*}'' \cup \hat{\mathcal{J}}_{i^*}''$ in the other. The effect of this branching rule on pricing is simply that of fixing some of the variables to a value of zero.

7.3.3 Columns management

Initialization. The initial RLMP is populated with several columns to allow for a "warm start" of column generation. For all problems outlined above, owing to the capacity constraint, even finding a feasible solution can be difficult, since the decision version of the problems is $\mathcal{NP}$-complete. However the initial columns must not necessarily form a complete feasible solution: they must only be in some way "reasonable", that is they must have a structure similar to those which are likely to be part of feasible and good solutions. Therefore they are generated in a heuristic way, as described in [18]: a given number p of candidate sites are selected at random and the customers are clustered around them. If the problem has a cardinality constraint, p is the given number of facilities to select; otherwise it is set to $|\mathcal{M}|$. In our implementation this initialization routine is run 10 times, generating 10 p columns. Furthermore we add the following dummy columns: first a dummy column having an entry equal to 1 corresponding to constraints (7.19) and 0's elsewhere; this represents a column covering all customers. This is done to ensure the existence of a feasible starting basis for each iteration of column generation. Second, we include a set of $|\mathcal{M}|$ dummy columns with an entry equal to 1 corresponding to only one constraint (7.15) and 0's elsewhere. This is done to ensure the existence of a feasible starting basis in each node of the branching tree when, due to branching or variable fixing operations, a site has been selected to host a facility. This second set of dummy columns can be dropped when the concentrator variants of the problems are used, since the first dummy column can be selected to fulfill such branching or fixing decisions.

Pool management. At each column generation iteration we insert into the RLMP all the columns with a negative reduced cost found by the pricing algorithm, if any. When a limit of 3000 columns in the RLMP is reached, we remove from the RLMP all the columns with a reduced cost larger than a given threshold; the threshold is set to the value of the ratio between the current primal-dual gap and the number of potential facilities p. Like in the initialization step, p is set to the maximum number of facilities to select: it is part of the input on the cardinality constrained problems, fixed to $|\mathcal{M}|$ otherwise. The removed columns are moved into a pool. At each column generation iteration we scan the pool; if any column is found to have a negative reduced cost, it is re-inserted into the RLMP. After three subsequent unsuccessful checks, the column is deleted also from the pool.

7.3.4 Lagrangean bound

The lower bound obtained from the LMP can also be obtained through the Lagrangean relaxation of semi-assignment constraints (7.2) and regional constraints (7.11) and (7.12) of the compact formulation:

$$
\begin{aligned}
LR)\ \min\ \ & \omega_{LR} = \sum_{j\in\mathcal{M}} f_j y_j + \sum_{i\in\mathcal{N}} d_{ij}x_{ij} + \\
& - \sum_{i\in\mathcal{N}} \lambda_i (\sum_{j\in\mathcal{M}} x_{ij} - 1) + \\
& - \sum_{R\in\mathcal{R}} \gamma_R^u (u_R - \sum_{j\in R} y_j) - \sum_{R\in\mathcal{R}} \gamma_R^l (\sum_{j\in R} y_j - l_R) \\
\text{s.t.}\ \ & \sum_{i\in\mathcal{N}} w_i x_{ij} \le Q_j y_j && \forall j \in \mathcal{M} \qquad (7.20) \\
& x_{ij} \in \{0,1\} && \forall i \in \mathcal{N}, \forall j \in \mathcal{M} \\
& y_j \in \{0,1\} && \forall j \in \mathcal{M}
\end{aligned}
$$

The Lagrangean multipliers in LR correspond to the dual variables in the LMP and the Lagrangean subproblem can be decomposed into the same M binary knapsack problems as the pricing subproblem in the column generation approach (for the equivalence between Dantzig-Wolfe decomposition and Lagrangean relaxation the reader is referred to mathematical programming textbooks like [77]). Therefore column generation can be used as a method alternative to subgradient optimization to update the Lagrangean multipliers.

At each iteration of column generation the current values of the dual variables $\boldsymbol{\lambda}, \boldsymbol{\gamma^u}$ and $\boldsymbol{\gamma^l}$ are used as Lagrangean multipliers. The optimal solution of the formulation above, that is a valid lower bound, can be computed as follows. First, we solve the allocation subproblem: for each candidate facility j, let a "penalty" value π_j be computed as

$$\pi_j = -f_j + \tau_j - \sum_{R \in \mathcal{R} | j \in R} (\gamma^u_R - \gamma^l_R)$$

where τ_j is the value of optimal solution of KP_j, that has been found solving the pricing problem. Second, we solve the location subproblem, that consists in finding the set $\mathcal{M}^{LR}$ of location sites which is feasible with respect to the regional constraints and contains the most profitable facilities (those with the lowest penalty values).

When regions do not overlap, $\mathcal{M}^{LR}$ can be computed as follows. Let $\underline{\mathcal{M}}^{LR}_R$ be the set of the l_R sites with maximum π_j values in region R and let $\bar{\mathcal{M}}^{LR}_R$ be the set of the u_R sites with maximum π_j values in region R. Then, the best selection of facilities can be computed in two steps. First the region lower bound constraints are satisfied by selecting the following set of facilities:

$$\mathcal{M}^{LR} := \bigcup_{R \in \mathcal{R}} \underline{\mathcal{M}}^{LR}_R$$

Second, the unselected facility

$$j^* \in \bigcup_{R \in \mathcal{R}} (\bar{\mathcal{M}}^{LR}_R \cap \underline{\mathcal{M}}^{LR}_R)$$

with highest π_{j^*} value is iteratively chosen, until the π_j value of the best facility left out is positive, or the upper bound constraints on the number of facilities in each region become tight. It is also easy to handle global upper and lower bounds on the number of facilities: the second step can be halted whenever one of the following conditions holds: (a) the global lower bound on the number of open facilities is satisfied and the π_j value of the best facility left out is positive; (b) the upper bound constraint on the number of facilities in each region becomes tight; (c) the global upper bound becomes tight.

Instead, when regions may overlap it is not easy to find the best set $\mathcal{M}^{LR}$. Therefore we further relax the problem and search for the most profitable set of facilities, considering only a subset of regions $\mathcal{S} \subseteq \mathcal{R}$ defined as follows. We start with $\mathcal{S} = \emptyset$. Then we iteratively choose the region $R \in \mathcal{R} \setminus \mathcal{S}$ of minimum cardinality with an empty intersection with each region in $\mathcal{S}$, and we include R in $\mathcal{S}$ until no more such regions can be found.

Once $\mathcal{S}$ has been found, we compute $\mathcal{M}^{LR}$ by solving the location subproblem as described above, replacing $\mathcal{R}$ with $\mathcal{S}$. That is, we drop the regional constraints in $\mathcal{R} \setminus \mathcal{S}$, obtaining the aforementioned relaxation, and we find the corresponding best set of location sites.

Thus, $\mathcal{M}^{LR}$ is the best set of location sites for either the Lagrangean problem or a relaxation of it, and a feasible bound is obtained as

$$\omega_{LR} = -\sum_{j \in \mathcal{M}^{LR}} \pi_j + \sum_{i \in \mathcal{N}} \lambda_i + \sum_{R \in \mathcal{R}} (\gamma^l_R l_R - \gamma^u_R u_R).$$

In this way a sequence of valid lower bounds is computed during column generation and this allows to fix variables or even to prune the current node of the search tree before column generation is over.

We further exploit the relationship between column generation and Lagrangean relaxation outlined above to improve the dual variables via subgradient optimization [47] after each column generation iteration. Starting with the current optimal values of the dual variables $\boldsymbol{\lambda}$, 100 subgradient iterations are executed. The step parameter is initialized at 2 and it is halved after every 10 iterations in which the current lower bound has not been improved with respect to the previous iteration.

Column generation is also speeded up by multiple pricing: instead of inserting into the RLMP only the optimal column for each candidate facility, if any is found with negative reduced cost, we add more (suboptimal) columns to enlarge the search space for the linear programming algorithm. This is particularly useful at the root node, when the column pool is still empty and the set of available columns may be small.

To this purpose we exploit the subgradient optimization algorithm and we insert into the RLMP the set of columns corresponding to each solution of the LR for which the Lagrangean lower bound improves upon the best incumbent Lagrangean lower bound.

7.3.5 Variable fixing

Given a solution of the Lagrangean relaxation LR, let ω_{LR} be its value and let

$$j^{WI}_R \in \text{argmin}_{j \in \mathcal{M}^{LR} \bigcap R} \{\pi_j\}$$

be the site in region R with minimum π_j value that hosts a facility and

$$j^{BO}_R \in \text{argmax}_{j \notin \mathcal{M}^{LR} \bigcap R} \{\pi_j\}$$

be the site in region R with maximum π_j value that does not host a facility (WI stands for "worst in", BO for "best out"). Let also v^* be a primal bound. The idea is to compute how forbidding the location of a facility in a site $j \in \mathcal{M}^{LR}$ would affect the dual bound. If $\pi_{j_R^{BO}} > 0$, or the lower bound constraint on region R is active (once building a facility in site j is forbidden, it is worth or necessary to open a facility in site j_R^{BO}), and $\lceil \omega_{LR} + \pi_j - \pi_{j_R^{BO}} \rceil \geq v^*$, then y_j can be fixed to 1. In a similar way, if the lower bound constraint is not active, $\pi_{j_R^{BO}} \leq 0$ and $\lceil \omega_{LR} + \pi_j \rceil \geq v^*$, then y_j can be fixed to 1. Analogously, if $\pi_{j_R^{WI}} < 0$ or the upper bound constraint on region R is active (once building a facility in j is imposed, it is worth or necessary to close the facility in site j_R^{WI}) and $\lceil \omega_{LR} - \pi_j + \pi_{j_R^{WI}} \rceil \geq v^*$, then y_j can be fixed to 0; in a similar way, if the upper bound constraint is not active, $\pi_{j_R^{WI}} \geq 0$ and $\lceil \omega_{LR} - \pi_j \rceil \geq v^*$, then y_j can be fixed to 0. Once the π_j values have been computed, this variable fixing step takes $O(\sum_{R \in \mathcal{R}} |R|)$ time and it may reduce the problem size considerably.

In our experiments variable fixing was done at each iteration of the subgradient optimization algorithm at the root node and only at the end of column generation at the other nodes in the search tree.

7.3.6 Primal heuristics

In order to find good feasible solutions early in the search tree, we integrated two primal heuristics in the main algorithm. Both of them are extensions of heuristics presented in [18] and consist of two phases: the selection of facility locations and the allocation of customers to the facilities.

The first one is a Lagrangean-based algorithm, based on the Lagrangean rlaxation presented in subsection 7.3.4: let $\mathcal{M}^{LR}$ be the set of sites in which a facility is activated in a Lagrangean-relaxed solution. We fix each location variable y_j to 1 if $j \in \mathcal{M}^{LR}$, to 0 otherwise. This selection of facility sites can violate some regional constraint. In this case the heuristic fails in identifying a feasible solution. Otherwise, we proceed to the allocation step as in [18]: when partitioning constraints are not violated, we use the same assignments which appear in the Lagrangean-relaxed solution, and the allocation of the other vertices is done as in the heuristic algorithm of Martello and Toth [73] (called MTHG by the authors) with desirability coefficients $f_{ij} = -d_{ij}$; the heuristic can fail during the second step too, since it can be impossible to find a feasible allocation pattern. However, this heuristic proved to be sufficiently fast and effective to be run at each evaluation of a Lagrangean bound, that is, several times for each column generation iteration.

Another primal bound, based on the set partitioning formulation, is computed with a rounding technique, starting from the current optimal solution of the LP relaxation of the master problem, described by the (fractional) variables z_k^j. In order to measure the desirability of building a facility in each site we define two sets of coefficients:

$$\phi_{ij} = \sum_{k \in Z^j} x_i^k z_k^j \quad \forall i \in \mathcal{N} \quad \forall j \in \mathcal{M}$$

and

$$\psi_j = \sum_{i \in \mathcal{N}} \phi_{ij} \quad \forall j \in \mathcal{M}.$$

Then, the selection of facility locations is done in two steps. In the first step the facility with highest ψ_j value is selected, among those which belong to some region whose lower bound is not satisfied yet, and do not belong to any region whose upper bound is tight; this operation is repeated until all region lower bounds are satisfied, or no such facility can be found. In the second case the heuristic fails in finding a feasible solution and stops. In the second step, the facility with highest ψ_j value is selected, among those which do not belong to a region whose upper bound is tight; this operation is repeated until no such facility can be found, or the ψ_j value of the selected facility is less than 0.5. For the allocation phase, we follow the three steps of the original MTHG algorithm: first, the customers are assigned to the facilities in decreasing order of desirability coefficients ϕ_{ij} as far as the capacity constraint allows the assignments; if some vertex remains unassigned, local search iterations are performed to produce a feasible solution; finally, if this step succeeds, a local search tries to improve the solution. This heuristic is much more time-consuming than the previous one. Therefore, it is used only in two cases: (a) at the root node, at each column generation iteration, provided that the value of the fractional LRMP solution is less than the double of the best known lower bound; (b) at each node of the search tree, only once the column generation process is over.

7.4 Branch-and-cut

The structure of a branch-and-cut algorithm is the following. First, the continuous relaxation of a compact formulation is solved. When the optimal solution of this relaxation is integral, it is also optimal. Otherwise, integrality is enforced in two ways, that is cutting planes and branching. Cutting planes are inequalities which are redundant for the original integer program, but are violated in the relaxed solution. Adding these cuts and re-optimizing the problem may yield a tighter lower bound. This cut-generation process can be iterated in order to obtain tighter approximations to the optimal integer solution. Branching is usually performed by selecting a variable whose value is fractional and considering two (or more) subproblems in which this variable is fixed to an integer value.

To benchmark our branch-and-price approach we chose as a competitor a general-purpose MIP solver, which uses branch-and-cut, that is ILOG CPLEX 8.1. In this section we review the classes of inequalities that CPLEX automatically generates and that we found to have a greater effect on the solution of single-source capacitated location problems.

Clique cuts. Clique cuts are added whenever a set of binary variables is identified such that at most one of the variables can be set to 1. These cuts are derived from the examination of the relationship between the variables through constraint propagation techniques. This is done by CPLEX in a preprocessing step, before optimization starts.

Minimal cover cuts and generalized upper bound cover cuts. Each capacity constraint (7.3) is analyzed in order to find a group of variables forming a minimal cover. A minimal cover is a set of variables such that if all of them were set to 1, the constraint would be violated, but if any of them is set to 0, the constraint would not be violated. Therefore a valid inequality, called cover inequality, imposes that the sum of these variables has to be strictly less than their number. These cover inequalities can be strenghtened in many ways. In particular CPLEX implements a search for generalized upper bound (GUB) cover cuts. A GUB imposes that at most one element in a subset of variables can be selected; this piece of information can be used to derive more restrictive cover cuts. CPLEX dynamically generates violated minimal cover cuts and GUB cover cuts, automatically finding how often to start this generation process and how many cuts to generate. These inequalities proved to be the most effective cuts for our class of integer programs.

Gomory fractional cuts. Gomory fractional cuts are an algebraic method for generating valid inequalities through integer rounding. This is a general purpose technique, that does not rely on any particular structure of the model. CPLEX allows the user to decide a number of parameters, including how many Gomory cuts must be generated and when. We kept the standard settings of CPLEX in our experiments since we observed they were quite effective.

7.5 Computational analysis

In this section we present the experimental results of our tests, in which we compared our general branch-and-price algorithm with the branch-and-cut-based solver of ILOG CPLEX 8.1.

We have divided our experiments into three different parts. The purpose of the experiments in the first part is to evaluate the effect of fixed costs, capacities and cardinality constraints on the computing time required to achieve a provably optimal solution. The second part concerns the effect of the introduction of regional constraints and concentrator models. The third part includes experiments on large-size instances for which neither approach could reach proven optimality within a time-out of several hours; the purpose of these last experiments is to compare branch-and-price with branch-and-cut in terms of approximation, measured by the primal-dual gap.

The branch-and-price algorithm has been implemented in C++. ILOG CPLEX 8.1 libraries have been used to solve the LP relaxations. The program was compiled with GNU C/C++ compiler version 3.2.2 with full optimizations. All internal parameters of CPLEX have been kept to default values. All tests have been run on a Pentium IV 1.6GHz machine, running a Linux RedHat 9 operating system. Resource limitations were imposed to both algorithms: computation was halted after one hour of CPU time and the available RAM memory was limited to 512 MB.

7.5.1 Cardinality constraints and fixed costs

This first set of experiments is aimed at studying the effect of cardinality constraints and fixed costs.

To this purpose we considered two SS-CFLP data-sets, both taken from the literature: the first one (indicated as HOLM) consists of 71 instances and it is described in [49]; the second (indicated as DIAZ) consists of 57 instances and it is described in [28]. The instances in these data-sets have up to 200 users and 30 candidate sites. DIAZ instances have non-uniform fixed costs and capacities, while HOLM instances consider candidate facilities with both uniform and non-uniform fixed costs and capacities

We designed our experiments to investigate two main questions: (Q1) *"What happens when a cardinality constraint is introduced?"* (Q2) *"How does the computing time change when fixed costs are made more uniform?"*

To answer question (Q1) we added a constraint on the maximum number of facilities, p. The value of p has been defined so that the average demand satisfied by each of p facilities would be equal to 0.8:

$$p = \lfloor \frac{\sum_{i \in \mathcal{N}} w_i}{0.8 \sum_{j \in \mathcal{M}} Q_j / |\mathcal{M}|} \rfloor.$$

The fixed costs are still considered.

To answer question (Q2) we also considered three different scenari: scenario (a) corresponds to the original HOLM and DIAZ instances with the additional cardinality constraint; in scenario (b) all fixed costs have been halved; in scenario (c) all fixed costs have been set to 0; for each DIAZ instance, in a forth scenario (d) the fixed cost for each site has been set equal to the average fixed cost in its instance.

In Tables 7.2 and 7.3 we report the results obtained for HOLM and DIAZ data-sets respectively.

Table 7.2 consists of four vertical blocks. The first one indicates the characteristics of each instance (in turn, problem id, number of users, number of candidate location sites, range in which the setup costs are generated, range in which the capacity constraints are generated); a single value substitutes the range when the data is constant. Each of the three subsequent blocks refers to one of the scenari (a) to (c) described above. In each of these blocks we report the value of the solutions found by the branch-and-price (BP) and branch-and-cut (BC) algorithms, and the CPU time spent in proving its optimality. When the test exceeded the time resource limitation, the corresponding 'time' column is marked with a dash, while in case of memory overflow we mark the 'time' column with a star symbol. When one (or both) method exceeded the resource limitations, and the best found solutions are different, we mark the one with lowest value in bold.

The last three rows of the table indicate the average computing times (neglecting the instances in which optimality was not proved), the number of instances solved to proven optimality and the average gap between the primal and the dual bounds at the end of computation (for the instances whose solutions were not proved optimal).

Table 7.3 has an analogous structure: in the first block we report the size of the instances. In the other blocks we report the results for scenari (a) to (d).

Question (Q1). In the rightmost block of each table we mark with a capital 'T' the instances for which the cardinality constraint is tight in an optimal solution. The leading row of each column is marked as the corresponding scenario; the column corresponding to scenario (c) has been dropped, since without fixed costs, the cardinality constraint is always tight. Considering scenario (a), the cardinality constraint is tight only for 6 of the 57 DIAZ instances, but has impact on many of the HOLM instances.

Question (Q2). From the analysis of the average results it is easy to see that non-uniform fixed costs make the instances much harder to solve for both branch-and-price and branch-and-cut. Computing times in scenario (c), without fixed costs, are two orders of magnitude lower than those in the other scenari and scenario (c) is the only one in which all DIAZ instances were solved within the time limit. In scenario (c) the branch-and-price algorithm could solve 63 HOLM instances out of 71 and CPLEX solved all of them; the increase in the average computing time of the branch-and-price algorithm with respect to scenari (a) and (b) is a consequence of the larger number of instances solved, since the average time is computed only on the solved instances.

Branch-and-price is more effective when fixed costs are absent or uniform: for instance in data-set DIAZ, scenario (c), it takes less average time than CPLEX to solve the same number of instances and in scenario (d) it takes about 75% the average time required by CPLEX and solves approximately the same number of instances. In data-set HOLM the number of instances closed by branch-and-price increases when costs become more uniform or vanish, while branch-and-cut is rather insensitive to this variation. When fixed costs are non-uniform branch-and-cut performed better than branch-and-price. This outcome was expected since our branch-and-price algorithm had been originally devised for the CPMP without fixed costs. When fixed costs are significant, location decisions (that is, where to open the facilities) are likely to become more critical than allocation decisions (that is, to which facility each user must be assigned). Hence in these cases a two-levels branching policy like that proposed by Pirkul [86] and Diaz and Fernandez[28] can be more appropriate for branch-and-price algorithms.

7.5.2 Regional constraints and concentrators

The purpose of the second set of experiments is to evaluate the effect of introducing cardinality constraints and fixed costs in SS-CFLP and CPMP models, from the viewpoint of the computational resources required to reach provable optimality and from the viewpoint of the optimal value. This completes the investigation addressed in Q1 and Q2: in the former case, we introduced cardinality constrains in instances involving fixed costs, while in this case we provide fixed costs in instances with cardinality restrictions. Hence the questions we investigated are the following: (Q3) *"What is the effect of introducing cardinality constraints in SS-CFLP?"* (Q4) *"What is the effect of introducing fixed costs in the CPMP?"*

Then we also tried to evaluate the effect of regional constraints and of constraints imposing that a user hosting a facility must be allocated to it (i.e. the "concentrator" case). We aim at giving an answer to the following questions: (Q5) *"What is the effect of introducing regional constraints of different types?"* (Q6) *"What is the effect of the constraint $y_i = x_{ii}$ in these models?"*

For this second set of experiments we used the data-set based on 20 CPMP instances taken from the OR Library web site and already used in several papers

([6] [84]). This data-set is more significant for these experiments for the following reasons: (1) they are Euclidean, in that distances between users and facilities are computed according to the Euclidean metric in two dimensions; therefore their distance matrices are more realistic than random matrices; (2) in the literature regional constraints have been so far considered in addition to models as CPMP [79] and SS-CFLP with cardinality constraints [99] and the instances considered in these papers are similar to ours; (3) the set of sites which can host facilities coincides with the set of the users; this restriction contributes to make these instances more realistic. Moreover it allows to evaluate the effect of the constraint $y_i = x_{ii}$.

The original instances have a cardinality constraint with $p = N/10$ and no fixed costs. We considered three variants: CPMP with cardinality constraint and no fixed costs, SS-CFLP with fixed costs and no cardinality constraint, and CARD+FIX with cardinality constraint and fixed costs. Fixed costs have been randomly generated from a uniform distribution in the range $[Q_j/2, \dots, 3Q_j/2]$.

We generated seven types of regional constraints, indicated by capital letters from A to G. Regions in types A to E correspond to partitions of the instance graph, while regions in type F and G instances may overlap. In Table 7.1 we describe in details how these instances were generated. We report the regions type (first column), the number of regions (second column), the ranges in which the lower and the upper bounds of each region are chosen (third and forth column) and the average overlapping of the regions (fifth column).

Let r be the number of regions in which a graph with N vertices has to be partitioned. Consider as "covered" a vertex belonging to a region: so each of the vertices in the original CPMP instances is initially "uncovered". In each instance regions are created as follows: first, the vertex with minimum average distance between the other vertices is selected. This is the "seed" of the new region. Second, the uncovered vertex with minimum distance between one of the vertices in the new region is iteratively included in the region, until the number of vertices in the region is N/r. This process is repeated, partitioning in $r - 1$ regions the graph of the uncovered nodes.

Regions overlapping is allowed by iteratively selecting the pair of vertices with minimum distance, which belong to different regions, and imposing that each of these vertices belongs to both regions, until a predefined overlapping ratio is reached.

The outcome of the experiments is reported in Tables 7.4 and 7.5. Each Table is divided in seven horizontal blocks and four vertical blocks. Each horizontal block corresponds to a region type (whose ID is reported in the first cell of each block). The first vertical block includes the instance parameters (region type and instance ID). The second, third and forth vertical blocks refer to the CPMP, SS-CFLP and CARD+FIX variants of the problem respectively. In each of these blocks

Constraint Type	Number of regions	Lower bound	Upper bound	Overlap percentage
A	1	0	p	0%
B	2	2	3	0%
C	$\lceil\frac{1}{2}p\rceil$	$[0,1]$	$[1\ldots3]$	0%
D	$\lceil\frac{2}{3}p\rceil$	$[0,1]$	$[1\ldots2]$	0%
E	p	1	1	0%
F	$\lceil\frac{3}{2}p\rceil$	$[0,1]$	$[1\ldots3]$	50%
G	$\frac{9}{5}p$	$[0,1]$	$[1\ldots3]$	80%

Table 7.1: Generation of regional constraints

we indicate, for each instance, the value of the optimal solution, the maximum distance between a facility and one of its assigned users and the maximum load of a facility (computed as the ratio between the sum of the demands of users assigned to the facility and the capacity of the facility). For each of these values, the percentage increase (or decrease) with respect to the value in the optimal solution of the problem without regional constraints is indicated. Finally, the time (in seconds) required to prove optimality by both branch-and-price and branch-and-cut is reported. In the last row of each horizontal block we indicate the average results for each region type.

Question (Q3). Comparing the results reported in block SS-CFLP with those in block CARD+FIX in Tables 7.4 and 7.5 it can be seen that the cardinality constraint has little effect on the computing time for both methods. Against the intuition, the introduction of this constraint does not contribute to reduce the search space and thus the difficulty of the problem: in some cases the opposite effect is observed. We remark that these experiments have been made only with the given value of the ratio $p/N = 1/10$. Also the effect on the value of the optimal solution was almost negligible: for 50 users instances we observed a 2.25% increase in the minimum cost, while for 100 users instances the observed increase was about 0.65%. Furthermore, this increase vanishes as tightest regional constraints are introduced.

Question (Q4). On the opposite, comparing the results in block CPMP with those in block CARD+FIX in Tables 7.4 and 7.5 it is clear that removing the fixed costs really changes the structure of the problem. For the branch-and-cut, the CPMP is more difficult than the CARD+FIX variant: computing the optimal solution of the CPMP instances requires about a double CPU time with respect to the variant with fixed costs. Branch-and-price does not have a regular behavior: on some instances (e.g. instance 8 with $N = 50$ and instance 8 with $N = 100$) the optimization of the CPMP version is more difficult than that of the CARD+FIX version, while on other instances the opposite trend is observed.

If CPMP is interpreted as a CARD+FIX problem in which obtaining the best average service for the users (minimum allocation costs) is much more relevant than searching for the best trade-off between fixed costs and allocation costs, then it makes sense to compare the optimal CPMP solution value with the contribution of the allocation costs to the optimal CARD+FIX solution value. When the CPMP model is used, a reduction of about 6% and 9.5% in the allocation costs is obtained for the $N = 50$ and $N = 100$ instances respectively, while the corresponding increase on the overall solution value is about 9.5% and 15.5% respectively.

Question (Q5). From a comparison of the values reported on the last row of each block in Tables 7.4 and 7.5 (the "avg" row), it is possible to evaluate the effect of the different regional constraints on the computing time and the optimal value. From the viewpoint of computational resources no relevant differences are observed: the computing time required by branch-and-price and branch-and-cut remains of the same order of magnitude.

In CPMP instances, with no fixed costs, when the number of regions increases (from A instances to E instances) the computing time decreases and this holds for both approaches. On the contrary in SS-CFLP instances, without cardinality constraints, the computing time increases when the number of regions increases. The computing time with overlapping regional constraints (types F and G) are not very different from those without regional constraints.

The value of the optimal solution is strongly affected by regional constraints: for instance the average optimal value of all the instances in class E is about 25% worse than that for class A. When regions are large and overlap (types F and G) the effect on the optimal value is small.

Instances with fixed costs, that is those in classes SS-CFLP and CARD+FIX, are more affected by regional constraints. This was expected because constraints on location variables have impact both on assignment costs and on location costs: in the former case, because it can be necessary to locate facilities in sites farther away from users; in the second case, because it can be necessary to use sites with higher fixed costs.

Since regional constraints are usually considered as an option to enforce some kind of "equity" in the geographical distribution of the facilities, we analyzed how the worst-case service varies when the regional constraints are introduced. In particular we observed the maximum user-facility distance and the maximum load assigned to a facility. Neither of these parameters significantly decreases when the number of regions increases. This puts some doubt on the actual effectiveness of regional constraints to achieve fairness among the users and equitable distribution of the workload among the facilities. It seems to us that equity can be better pursued by suitable models in which min-max objective functions are explicitly considered and optimized.

Question (Q6). All tests described in the previous paragraph have been repeated for the concentrator variant, in which user i must be assigned to facility j whenever the facility is located in the same site of the user. Results are reported in Tables 7.6 and 7.7.

We can observe that the average computing time of branch-and-cut is improved; this was also expected since branch-and-cut takes advantage of a reduced number of variables. On the contrary the computing time of branch-and-price is worse, even if the N constraints (7.15) are removed from formulation (7.13) – (7.18).

The value of the optimal solution is not affected: it did not change in any of our tests.

7.5.3 Lower bounds and gaps on large-size instances

In former computational evaluations [18], the lower bounds achieved by branch-and-cut and branch-and-price at the root node have been measured, when solving large scale instances (900 customers). We observed that the lower bound provided by branch-and-price is consistently tighter: branch-and-cut could never reduce the duality gap with respect to the best known feasible solution below 1.2%; on the contrary branch-and-price achieved duality gaps ranging from the 0.5 to the 0.8 as much. CPLEX experienced memory overflow problems and could neither solve the linear relaxation at the root node nor find any feasible solution. On the contrary branch-and-price yielded feasible solutions with an approximation of about 4% in average.

On the same large-scale instances, we observed that when the concentrator restriction is introduced, the computing time required by branch-and-price is reduced to approximately one half.

7.6 Conclusions

From the average results reported in the last rows of the tables above the following observations can be done. Both branch-and-cut and branch-and-price could solve all instances with 50 users: branch-and-cut was in average faster; branch-and-price worked better when both cardinality constraints and fixed costs were present (class CARD+FIX). For instances with 100 users branch-and-cut was clearly superior to branch-and-price, mainly in terms of instances solved to proven optimality within the time limit. Looking at the detailed results reported in the tables, however, one can see that there are several instances in which branch-and-price wins. Therefore the superiority of branch-and-cut holds in average, on a sufficiently large number of instances, but there is no guarantee that it will be the best approach on any given single instance.

A very important remark concerns the trade-off between the width of applicability and the effectiveness of the algorithms examined here. As stated in the introduction, our goal was not to compare two specific algorithms but rather two approaches, designed to be widely applicable to all location problems considered here. This strive for generality is obviously paid in terms of effectiveness. Therefore for each single location problem considered here it may be possible to obtain better results than those presented above, by incorporating specialized cutting, branching and heuristic procedures. Examples of specialized cutting planes devised for particular network location problems are those of [3] and [2]; specialized branching procedures are illustrated for instance in [28] and [86]; specialized heuristic procedures can be found in [49] and [29]. The performances of both branch-and-cut and branch-and-price general solvers can be strongly affected by such tailored additions. Branch-and-price exploits a tight lower bound, provided by the set partitioning reformulation of the problems. This gives advantage on large scale instances compared with branch-and-cut algorithms.

Last but not least, the branch-and-cut code we have used as a benchmark for our branch-and-price framework is a commercial Mixed-Integer Linear Programming solver, not available for free, while our code is freely available upon request for scientific purposes. It needs a linear programming subroutine to solve the linear relaxation of the restricted master problem: the results presented above have been obtained with the ILOG CPLEX simplex algorithm but any linear programming solver can be used instead.

					(a) Setup * 1.0, load factor 0.8				(b) Setup * 0.5, load factor 0.8				(c) Setup * 0.0, load factor 0.8				Tight cons.	
Id	N	M	F	Cap	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	(a)	(b)
1	50	10	[300..700]	[100..400]	8848	0.38	8848	0.17	7331	0.44	7331	0.17	5814	0.27	5814	0.16	T	T
2			300		7914	0.38	7914	0.15	6864	0.27	6864	0.16	5814	0.26	5814	0.16	T	T
3			500		9314	0.30	9314	0.15	7564	0.34	7564	0.16	5814	0.26	5814	0.15	T	T
4			700		10714	1.53	10714	0.74	8264	0.26	8264	0.15	5814	0.26	5814	0.14	T	T
5			[300..700]	200	8838	0.41	8838	0.40	7026	0.38	7026	0.12	5077	0.41	5077	0.12		T
6			300		7777	0.20	7777	0.06	6427	0.46	6427	0.11	5077	0.19	5077	0.05	T	T
7			500		9488	1.32	9488	0.28	7327	0.54	7327	0.12	5077	0.40	5077	0.12		T
8			700		11088	1.10	11088	0.45	8227	1.05	8227	0.19	5077	0.40	5077	0.12		T
9			[300..700]	300	8462	0.30	8462	0.14	7197	0.36	7197	0.14	5932	0.32	5932	0.14	T	T
10			300		7732	0.38	7732	0.14	6832	0.35	6832	0.14	5932	0.31	5932	0.14	T	T
11			500		8932	0.52	8932	0.14	7432	0.42	7432	0.14	5932	0.30	5932	0.14	T	T
12			700		10132	1.03	10132	0.15	8032	0.34	8032	0.14	5932	0.32	5932	0.14	T	T
13	50	20	[300..700]	[100..400]	8252	0.40	8252	0.32	6382	0.50	6382	0.26	4437	0.34	4437	0.23	T	T
14			300		7137	0.48	7137	0.25	5787	0.67	5787	0.22	4437	0.34	4437	0.23	T	T
15			500		8808	0.56	8808	0.28	6687	0.37	6687	0.24	4437	0.33	4437	0.24		T
16			700		10408	1.25	10408	0.57	7587	0.41	7587	0.26	4437	0.32	4437	0.23		T
17			[300..700]	200	8227	0.93	8227	0.44	6370	0.32	6370	0.23	4425	0.53	4425	0.23		T
18			300		7125	0.37	7125	0.23	5775	0.48	5775	0.23	4425	0.54	4425	0.24	T	T
19			500		8886	2.09	8886	1.43	6675	0.27	6675	0.24	4425	0.55	4425	0.22		T
20			700		10486	2.17	10486	1.44	7575	0.32	7575	0.27	4425	0.54	4425	0.24		T
21			[300..700]	300	8171	0.76	8171	0.18	6785	0.84	6785	0.22	5373	3.14	5373	2.13	T	T
22			300		7473	4.50	7473	2.54	6423	2.94	6423	1.95	5373	3.14	5373	2.17	T	T
23			500		8873	3.41	8873	2.23	7123	2.30	7123	0.61	5373	3.09	5373	2.14	T	T
24			700		10273	2.99	10273	1.66	7823	2.18	7823	0.46	5373	3.10	5373	2.04	T	T
25	150	30	[300..700]	[100..400]	11630	-	11630	15.62	9978	682.60	9978	1.15	8086	31.01	8086	0.98		T
26			300		10771	-	10771	15.27	9436	54.79	9436	0.98	8086	30.72	8086	0.97		T
27			500		12322	-	12322	43.97	10336	1594.03	10336	2.66	8086	31.01	8086	0.97		T
28			700		13722	-	13722	43.55	11171	-	11171	9.23	8086	30.69	8086	0.96		
29			[300..700]	200	12371	-	12371	226.02	10257	-	10257	130.12	7967	2690.72	7967	4.65		
30			300		11394	-	**11331**	670.97	9744	-	**9742**	46.10	7967	2697.21	7967	4.66		
31			500		13483	-	**13331**	546.72	10844	-	**10831**	273.28	7967	2682.70	7967	4.63		
32			700		15450	-	**15331**	1040.26	11944	-	**11831**	763.61	7967	2687.10	7967	4.64		
33			[300..700]	300	11629	512.66	11629	3.22	9932	498.81	9932	1.51	8068	31.36	8068	1.06		
34			300		10632	40.10	10632	1.33	9418	102.52	9418	1.27	8068	31.13	8068	1.06		T
35			500		12232	29.18	12232	1.34	10232	25.66	10232	1.32	8068	31.29	8068	1.06		
36			700		13832	31.97	13832	1.35	11032	39.33	11032	1.33	8068	31.10	8068	1.06		
37			[300..700]	600	11258	34.80	11258	1.03	10049	39.92	10049	0.99	8824	74.40	8824	1.02	T	T
38			300		10624	44.66	10624	0.95	9724	36.20	9724	0.98	8824	74.88	8824	0.99	T	T
39			500		11824	51.50	11824	1.02	10324	44.59	10324	0.95	8824	75.36	8824	0.99	T	T
40			700		13024	28.89	13024	0.97	10924	38.64	10924	0.99	8824	74.80	8824	1.01	T	T

Computational results on HOLM instances (cont'd on the next page)

					(a) Setup * 1.0, load factor 0.8				(b) Setup * 0.5, load factor 0.8				(c) Setup * 0.0, load factor 0.8				Tight cons.	
Id	N	M	F	Cap	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	(a)	(b)
41	90	10	[300..700]	[100..400]	6700	76.08	6700	18.14	5728	29.09	5728	8.90	4730	79.82	4730	8.30	T	T
42	80	20	[300..700]	[100..400]	5663	100.83	5663	6.90	4666	23.80	4666	6.33	3669	21.64	3669	2.63	T	T
43	70	30	[300..700]	[100..400]	5214	1.01	5214	0.55	4292	1.32	4292	0.55	3368	1.62	3368	0.54	T	T
44	90	10	[300..700]	[100..400]	7270	8.06	7270	2.08	5829	21.17	5829	0.57	4388	3.33	4388	0.30	T	T
45	80	20	[300..700]	[100..400]	6251	4.34	6251	0.49	5018	3.63	5018	0.50	3785	3.32	3785	0.53	T	T
46	70	30	[300..700]	[100..400]	5965	2.48	5965	0.78	4949	1.31	4949	0.79	3919	2.78	3919	0.80	T	T
47	90	10	[300..700]	[100..400]	6719	1598.23	6719	3.16	5354	65.78	5354	4.43	3975	41.87	3975	0.85	T	T
48	80	20	[300..700]	[100..400]	6179	29.11	6179	9.20	5017	19.06	5017	9.54	3857	20.75	3857	11.75	T	T
49	70	30	[300..700]	[100..400]	5609	61.82	5609	13.68	4529	5.68	4529	0.96	3417	3.41	3417	0.58	T	T
50	100	10	[300..700]	[100..400]	8808	-	8808	6.44	7654	-	7654	6.63	6500	-	6500	11.29	T	T
51		20	[300..700]	[100..400]	7422	-	**7414**	40.35	6219	-	6219	9.71	4971	2475.37	4971	14.56	T	T
52		10	[300..700]	[100..400]	9178	16.32	9178	1.47	7785	39.61	7785	1.51	6390	36.24	6390	1.00	T	T
53		20	[300..700]	[100..400]	8531	6.63	8531	0.58	7151	12.04	7151	0.59	5770	9.99	5770	0.57	T	T
54		10	[300..700]	[100..400]	8777	4.39	8777	0.95	7416	62.76	7416	1.19	5861	13.41	5861	1.10		
55		20	[300..700]	[100..400]	7654	210.13	7654	2.62	6324	73.49	6324	2.49	4894	13.29	4894	2.23		T
56	200	30	[300..700]	[100..400]	21120	-	**21103**	119.79	16620	-	**16618**	130.69	12178	-	**12118**	104.75	T	T
57			300		26411	-	**26039**	939.41	19318	-	19318	127.78	12178	-	**12118**	104.94		T
58			500		37871	-	**37239**	796.17	25720	-	**25239**	611.74	12178	-	**12118**	104.88		
59			700		27311	-	**27282**	341.88	20519	1434.28	20519	27.72	12178	-	**12118**	104.66		
60			[300..700]	200	20854	92.33	20854	1.88	17854	51.74	17854	1.89	14854	107.52	14854	1.89	T	T
61			300		24454	42.29	24454	2.20	19654	45.97	19654	1.95	14854	107.05	14854	1.90	T	T
62			500		32689	-	**32643**	193.20	23854	37.10	23854	2.08	14854	107.71	14854	1.91		T
63			700		25105	525.87	25105	6.84	20701	56.66	20701	2.50	14854	107.46	14854	1.91	T	T
64			[300..700]	300	22476	-	22476	68.19	20226	2891.49	20226	52.19	17976	-	17976	64.74	T	T
65			300		25176	2326.14	25176	59.09	21576	2266.30	21576	59.30	17976	-	17976	64.63	T	T
66			500		31657	-	**31415**	330.87	24726	-	24726	73.22	17976	-	17976	63.69	T	T
67			700		24848	422.25	24848	14.67	20248	32.85	20248	0.92	14619	37.51	14619	0.87		T
68			[300..700]	600	20932	17.19	20932	1.13	17932	55.73	17932	1.10	14932	18.91	14932	1.10	T	T
69			300		24532	15.47	24532	1.11	19732	20.84	19732	1.17	14932	18.86	14932	1.09	T	T
70			500		32392	-	**32321**	135.92	23932	39.66	23932	2.27	14932	39.90	14932	2.28		T
71			700		**25880**	-	25893	-	21019	-	**20973**	270.53	14932	39.98	14932	2.32	T	T
Avg computing time						122.36		82.11		177.38		37.54		232.33		10.24		
Solved instances						52.00		70.00		59.00		71.00		63.00		71.00		
Avg. (PB - DB) / DB gap						0.87%				0.80%				0.72%				

Table 7.2: Computational results on HOLM instances

			(a) Setup * 1.0, load factor = 0.8				(b) Setup * 0.5, load factor = 0.8				(c) Setup * 0.0, load factor = 0.8				(d) AVGFIX				Tight cons.	
Id	N	M	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	(a)	(b)
1	15	10	2014	2.51	2014	1.42	1147	0.30	1147	0.33	207	0.07	207	0.08	2081	0.08	2081	0.13		T
2			4251	0.63	4251	1.70	2319	2.11	2319	1.46	201	0.07	201	0.04	3668	0.21	3668	1.30		
3			6051	-	6051	2.40	3159	1.47	3159	0.86	185	0.05	185	0.03	5427	0.12	5427	0.12		T
4			7168	-	7168	10.91	3785	506.69	3785	1.05	195	0.06	195	0.03	6328	0.18	6328	0.14		
5			4551	-	4551	9.24	2466	3.97	2466	1.23	269	0.06	269	0.04	4273	0.14	4273	0.16		
6			2269	1.81	2269	1.67	1272	0.11	1272	0.24	213	0.03	213	0.03	2189	0.11	2189	0.17		
7	30	15	4366	1201.23	4366	8.96	2422	45.49	2422	10.25	269	0.06	269	0.04	4069	0.44	4069	2.82		
8			1244	0.06	1244	0.03	1231	0.05	1231	0.04	381	0.13	381	0.11	2216	0.05	2216	0.04		
9			2480	1.33	2480	19.22	1447	1.93	1447	7.99	275	0.09	275	0.08	2664	0.57	2664	14.58		
10			23112	2102.66	23112	4.40	11745	1.37	11745	20.82	172	0.08	172	0.05	21733	0.37	21733	19.75		
11			3447	7.53	3447	59.36	1937	0.33	1937	5.34	255	0.10	255	0.06	3245	0.58	3245	139.32		
12			3711	440.25	3711	39.34	2103	0.73	2103	4.60	297	0.08	297	0.08	3394	0.33	3394	0.62		
13			3760	335.45	3760	178.79	2090	0.97	2090	5.78	288	0.23	288	0.19	3380	1.52	3380	11.07		
14			6767	-	**6579**	4.43	3654	2220.36*	**3572**	122.26	323	0.08	323	0.06	6571	0.24	6571	0.24	T	T
15			7816	47.29	7816	21.93	4095	1.18	4095	28.08	175	0.11	175	0.08	7868	0.44	7868	9.30		
16			11543	-	11543	16.71	5958	5.96	5958	28.27	175	0.10	175	0.07	11648	0.39	11648	7.21		
17			9932	2025.03*	9884	18.93	5239	-	5239	11.56	243	0.07	243	0.08	9054	0.65	9054	1.90		
18	40	20	15628	-	**15607**	68.21	8037	150.39	8037	35.43	238	0.11	238	0.11	14427	1.28	14427	555.28		
19			18769	1506.45*	**18683**	2535.92	9596	-	**9557**	4.14	221	0.13	221	0.18	17301	1.49	17301	13.73		
20			26680	-	**26584**	-	13558	-	**13504**	1053.73	185	0.19	185	0.11	24862	2.65	24862	604.14		
21			7302	-	**7301**	-	3872	10.39	3872	400.90	265	0.19	265	0.15	6937	0.43	6937	26.11		
22			3315	-	**3314**	119.82	1926	4.78	1926	22.85	328	0.30	328	0.29	3166	64.81	3166	5.16	T	T
23			6036	118.78	6036	28.22	3338	0.84	3338	10.61	320	0.22	320	0.25	6160	1.11	6160	2.26	T	T
24			6357	2683.41*	**6346**	14.73	3474	-	3474	14.57	407	0.71	407	1.96	6550	100.17	6550	7.46	T	T
25			8947	-	8947	14.87	4745	-	4745	8.08	359	3.19	359	1.91	10024	29.35	10024	5.44		
26	50	20	4448	9.92	4448	365.19	2502	14.21	2502	384.02	268	0.19	268	0.22	4347	2.04	4347	12.97		
27			10974	-	**10963**	-	5981	-	**5979**	-	455	4.28	455	3.42	11734	81.82	11734	5.38	T	T
28			11590	-	**11500**	63.49	6080	-	**6025**	149.40	351	0.64	351	3.04	11327	478.67	11327	26.09	T	T
29			9941	-	**9832**	104.28	5239	-	5239	234.44	402	0.25	402	0.18	9662	224.95	9662	17.01		
30			**10935**	-	11074	-	5735	2424.67*	**5721**	-	249	0.35	249	0.14	9497	0.54	9497	8.33		
31			4525	2271.85*	**4466**	2864.39	2466	70.30	2466	17.81	285	0.21	285	0.12	3789	26.47	3789	20.39		
32			10266	1385.73*	**9881**	37.29	5295	-	5295	354.32	387	1.57	387	5.52	8965	19.66	8965	2.71		T
33			**39463**	-	41362	-	**20026**	1468.96	20943	-	311	0.89	311	0.33	38010	1.55	38010	740.29		

Computational results on DIAZ instances (cont'd on the next page)

			(a) Setup * 1.0, load factor = 0.8				(b) Setup * 0.5, load factor = 0.8				(c) Setup * 0.0, load factor = 0.8				(d) AVGFIX				Tight cons.	
Id	N	M	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	BPv.	BPt.	BCv.	BCt.	(a)	(b)
34	60	30	4819	-	**4736**	-	2796	-	**2769**	527.66	315	0.30	315	0.37	8330	2900.03	8330	187.95		
35			5456	-	5456	26.82	2999	61.11	2999	21.20	256	1.06	256	1.33	8184	26.98	8184	459.94		
36			17243	-	**16781**	2008.69	8939	-	**8749**	2340.74	244	0.25	244	0.22	27416	7.44	27416	48.28		
37			**14783**	-	14803	-	7771	2742.9*	**7692**	-	262	0.99	262	6.04	23147	4.51	23147	34.45		
38			47280	-	**47249**	112.95	**23863**	-	23875	-	199	0.30	199	0.44	54037	2.86	54037	142.28		
39			41197	-	**41007**	856.11	20851	-	**20766**	3111.67	233	0.26	233	0.25	55083	8.67	55083	535.18		
40			**62004**	-	64108	-	31267	-	**31056**	-	183	0.71	183	0.98	73969	2.44	73969	465.55		
41			20202	462.65*	17246	17.73	10551	815.93*	**9196**	45.63	506	2.00	506	13.43	60052	-	60052	46.30		
42	75	30	7920	-	**7887**	159.80	4366	-	**4344**	-	318	0.42	318	0.38	11070	205.36	11070	903.45		
43			5114	3.00	5114	499.35	3020	-	3020	38.69	378	9.49	378	13.43	10274	8.52	10274	45.89		
44			**36688**	-	37636	-	**18389**	-	18959	-	211	0.55	211	0.29	38065	5.25	38065	211.58		
45			17854	-	**17676**	18.23	9308	-	**9233**	135.81	335	1.01	335	1.64	29359	63.67	29359	190.55		
46			51409	-	**50250**	-	26078	-	**24718**	2676.39	348	0.42	348	0.42	71251	70.66	71251	131.01		
47			**68426**	-	68452	-	**34603**	-	34995	-	291	0.43	291	0.31	67682	3.79	67682	416.88		
48			63385	-	**58964**	314.92	32181	-	**30036**	130.99	423	1.37	423	8.37	85726	1053.53	85729	1408.68		
49			85668	-	**81301**	-	42803	-	**41184**	-	285	0.45	285	0.57	92208	95.09	92208	1325.22		
50	90	30	6047	-	**5937**	248.99	3570	-	**3522**	643.45	550	14.38	550	15.47	12720	-	**12712**	66.75		
51			9123	3277.43	9123	-	**5053**	-	5055	-	369	0.70	369	0.68	12946	466.02	12946	301.92		
52			38575	-	**35324**	-	19572	-	**17681**	-	357	1.78	357	1.30	47157	30.77	47157	245.88		
53			30140	-	**30038**	339.95	15558	-	**15379**	808.02	322	0.88	322	0.41	37580	9.57	37580	1639.23		
54			51461	-	**43853**	39.32	26203	-	**22485**	59.48	462	15.70	462	12.22	84562	37.89	84562	35.64		
55			**71009**	-	72034	-	**35849**	-	36422	-	351	0.76	351	0.44	**84255**	-	84313	-		
56			64474	-	64474	118.36	32759	-	**32755**	127.05	474	2.18	474	6.58	110009	2240.50	110009	82.10		
57			49791	1605.43	49791	6.80	25456	2239.79	25456	5.48	560	3.77	560	10.10	107410	-	107410	323.92		
Avg comp.t.				565.76		271.04		301.31		309.38		1.32		2.01		156.36		205.54		
Solved inst.				16.00		42.00		24.00		44.00		57.00		57.00		53.00		56.00		
Avg(P-D)/Dgap				6.70%				9.05%				0.00%				2.14%				

Table 7.3: Computational results on DIAZ instances

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
A	1	713	0.00%	3.58	4.29	1217	0.00%	0.65	0.27	1242	0.00%	5.94	3.84
	2	740	0.00%	0.36	0.28	1169	0.00%	0.68	0.27	1169	0.00%	0.24	0.25
	3	751	0.00%	1.41	1.33	1141	0.00%	1.17	0.39	1162	0.00%	0.36	0.5
	4	651	0.00%	0.46	0.56	1141	0.00%	0.89	0.55	1141	0.00%	0.37	0.41
	5	664	0.00%	0.94	0.75	1103	0.00%	4.24	0.44	1103	0.00%	0.45	0.38
	6	778	0.00%	0.54	0.67	1240	0.00%	1.33	0.44	1257	0.00%	5.73	16.39
	7	787	0.00%	5.08	17.14	1175	0.00%	1.64	1.93	1213	0.00%	1.71	6.45
	8	820	0.00%	1891.39	207.91	1184	0.00%	1.84	0.39	1248	0.00%	4.89	8.14
	9	715	0.00%	1.92	10.5	1141	0.00%	2.31	0.37	1152	0.00%	1.25	0.74
	10	829	0.00%	10.81	84.3	1159	0.00%	0.75	0.28	1248	0.00%	39.25	49.57
A	Avg		0.00%	191.65	32.77		0.00%	1.55	0.53		0.00%	6.02	8.67
B	1	713	0.00%	3.18	3.65	1217	0.00%	0.29	0.27	1242	0.00%	5.43	2.44
	2	740	0.00%	0.4	0.29	1169	0.00%	0.26	0.26	1169	0.00%	0.23	0.26
	3	751	0.00%	1.29	5.65	1150	0.79%	1.13	0.49	1162	0.00%	0.37	0.51
	4	651	0.00%	0.56	0.56	1141	0.00%	0.4	0.92	1141	0.00%	0.53	0.47
	5	664	0.00%	0.84	0.79	1103	0.00%	4.61	0.43	1103	0.00%	0.37	0.34
	6	778	0.00%	0.54	0.77	1240	0.00%	1.05	0.45	1257	0.00%	4.9	11.19
	7	787	0.00%	4.69	22.73	1175	0.00%	0.64	0.61	1213	0.00%	1.7	10.86
	8	821	0.12%	2559.2	219.64	1190	0.51%	0.74	0.64	1248	0.00%	4.32	7.28
	9	715	0.00%	1.36	9.56	1141	0.00%	1.11	0.37	1152	0.00%	1.19	0.79
	10	829	0.00%	11.09	77.86	1183	2.07%	0.98	0.69	1248	0.00%	21.28	28.88
B	Avg		0.01%	258.32	34.15		0.34%	1.12	0.51		0.00%	4.03	6.30
C	1	713	0.00%	2.01	1.26	1258	3.37%	1.34	0.61	1258	1.29%	0.57	0.44
	2	761	2.84%	1.05	0.38	1216	4.02%	1.35	0.63	1216	4.02%	1.65	0.61
	3	751	0.00%	1.12	1.16	1155	1.23%	1.01	0.62	1162	0.00%	0.49	0.64
	4	652	0.15%	0.81	0.41	1141	0.00%	0.49	0.47	1141	0.00%	0.4	0.51
	5	664	0.00%	0.64	0.92	1103	0.00%	5.75	0.98	1103	0.00%	0.59	0.47
	6	799	2.70%	0.76	0.84	1274	2.74%	7.1	14.2	1274	1.35%	1.7	11.89
	7	787	0.00%	5.45	21.38	1175	0.00%	1.55	0.43	1213	0.00%	2.03	7.39
	8	820	0.00%	1843.11	205.44	1195	0.93%	0.5	0.32	1248	0.00%	3.02	13.08
	9	715	0.00%	0.85	0.69	1152	0.96%	1.57	0.72	1152	0.00%	1.87	0.61
	10	835	0.72%	23.33	79.78	1183	2.07%	0.39	0.35	1248	0.00%	22.55	46.35
C	Avg		0.64%	187.91	31.23		1.53%	2.11	1.93		0.67%	3.49	8.20
D	1	744	4.35%	3.45	2.26	1270	4.35%	0.43	0.39	1270	2.25%	0.42	0.38
	2	770	4.05%	1.91	0.93	1209	3.42%	2.15	0.81	1209	3.42%	1.82	1.2
	3	755	0.53%	3.15	1.72	1186	3.94%	2	2.02	1186	2.07%	3.96	0.72
	4	651	0.00%	1.16	0.57	1163	1.93%	0.73	0.5	1165	2.10%	0.78	0.47
	5	664	0.00%	0.5	0.64	1103	0.00%	0.58	0.32	1103	0.00%	0.56	0.31
	6	778	0.00%	0.67	0.7	1240	0.00%	1.44	0.44	1257	0.00%	3.86	10.79
	7	845	7.37%	8.2	20.93	1315	11.91%	54.87	17.3	1315	8.41%	30.29	21.86
	8	820	0.00%	41.67	32.87	1285	8.53%	1.8	5.96	1285	2.96%	1.84	6.25
	9	717	0.28%	3.02	9.29	1146	0.44%	0.4	0.39	1165	1.13%	1.48	13.5
	10	848	2.29%	52.83	60.16	1217	5.00%	0.53	0.52	1253	0.40%	7.27	18.2
D	Avg		1.89%	11.66	13.01		3.95%	6.49	2.87		2.27%	5.23	7.37

Computational results on SSCFLP instances, N = 50 (cont'd on the next page)

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
E	1	714	0.14%	2.35	0.64	1266	4.03%	2.01	4.39	1266	1.93%	1.68	4.44
	2	740	0.00%	0.65	0.26	1170	0.09%	0.21	0.24	1170	0.09%	0.22	0.25
	3	751	0.00%	1.1	0.83	1162	1.84%	0.6	0.38	1162	0.00%	0.49	0.37
	4	652	0.15%	0.58	0.38	1141	0.00%	0.45	0.43	1141	0.00%	0.61	0.36
	5	664	0.00%	1.5	0.69	1103	0.00%	0.55	0.31	1103	0.00%	0.38	0.31
	6	787	1.16%	2.72	0.56	1274	2.74%	1.93	12.05	1274	1.35%	2.53	14.78
	7	789	0.25%	2.74	12.14	1255	6.81%	16.66	17.32	1255	3.46%	26.15	14.93
	8	822	0.24%	507.72	41.87	1266	6.93%	1.62	4.97	1266	1.44%	1.38	7.14
	9	718	0.42%	3.51	12	1183	3.68%	1.83	6.45	1183	2.69%	1.83	7.78
	10	829	0.00%	2.03	46.01	1277	10.18%	62.53	28.94	1277	2.32%	46.9	30.03
E	Avg		0.24%	52.49	11.54		3.63%	8.84	7.55		1.33%	8.22	8.04
F	1	713	0.00%	3.51	3.74	1217	0.00%	0.81	0.24	1244	0.16%	2.68	3.41
	2	750	1.35%	0.79	0.65	1181	1.03%	1.11	0.26	1181	1.03%	0.43	0.28
	3	755	0.53%	3.05	1.19	1148	0.61%	1.59	0.35	1177	1.29%	2.39	5.62
	4	651	0.00%	0.37	0.46	1141	0.00%	0.65	0.57	1141	0.00%	0.7	0.45
	5	666	0.30%	1.01	0.76	1161	5.26%	3.25	0.73	1161	5.26%	0.72	0.94
	6	778	0.00%	0.45	0.68	1248	0.65%	3.3	3.5	1257	0.00%	2.17	8.33
	7	787	0.00%	3.78	17.21	1179	0.34%	1.4	0.8	1232	1.57%	17.54	24.94
	8	820	0.00%	1726.75	105.1	1195	0.93%	0.88	0.3	1248	0.00%	3.79	10.93
	9	715	0.00%	1.73	7.78	1145	0.35%	0.67	0.29	1156	0.35%	1.07	0.56
	10	829	0.00%	5.89	48.53	1188	2.50%	1.79	2.16	1261	1.04%	11.09	34.95
F	Avg		0.22%	174.73	18.61		1.17%	1.55	0.92		1.07%	4.26	9.04
G	1	714	0.14%	1.6	1.42	1217	0.00%	0.51	0.26	1244	0.16%	6.57	3.09
	2	740	0.00%	0.57	0.3	1181	1.03%	0.7	0.22	1181	1.03%	0.65	0.24
	3	751	0.00%	0.62	0.38	1152	0.96%	1.36	0.29	1177	1.29%	0.49	0.37
	4	652	0.15%	0.55	0.42	1141	0.00%	1.04	0.57	1141	0.00%	0.65	0.43
	5	664	0.00%	1.37	0.71	1103	0.00%	0.76	0.3	1103	0.00%	0.67	0.28
	6	778	0.00%	0.7	0.71	1240	0.00%	1.62	0.42	1257	0.00%	2.13	6.16
	7	805	2.29%	4.5	25.5	1197	1.87%	1.4	0.57	1274	5.03%	9.97	21.54
	8	829	1.10%	1476.55	149.64	1232	4.05%	0.65	0.57	1267	1.52%	6.62	17.29
	9	715	0.00%	2.05	11.76	1141	0.00%	0.59	0.31	1171	1.65%	0.55	0.42
	10	829	0.00%	3.58	74.21	1183	2.07%	1.54	0.55	1289	3.29%	345.39	43.35
G	Avg		0.37%	149.21	26.51		1.00%	1.02	0.41		1.40%	37.37	9.32
Solved instances				70.00	70.00			70.00	70.00			70.00	70.00
Avg. computing time				146.57	23.97			3.24	2.10			9.80	8.13

Table 7.4: Computational results on SSCFLP instances, N = 50

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
A	1	1006	0.00%	16.87	66.5	1819	0.00%	9	3.08	1835	0.00%	20.47	69.29
	2	966	0.00%	106.59	96.01	1848	0.00%	62.34	23.58	1848	0.00%	50.72	34.07
	3	1026	0.00%	15.53	16.48	1834	0.00%	6.24	2.75	1842	0.00%	6.82	31.51
	4	982	0.00%	341.49	326.52	1845	0.00%	52.06	26.16	1850	0.00%	194.37	63.89
	5	1091	0.00%	279.72	273.44	1876	0.00%	52.46	16.71	1897	0.00%	397.41	227.42
	6	954	0.00%	9.54	138.49	1732	0.00%	12.37	2.3	1740	0.00%	7.33	7.35
	7	1034	0.00%	91.62	153.59	1834	0.00%	53.56	21.59	1851	0.00%	1853.16	177.35
	8	1043	0.00%	698.39	304.22	1812	0.00%	36.58	10.61	1812	0.00%	27.26	38.58
	9	1031	0.00%	36.45	185.51	1854	0.00%	239.58	39	1854	0.00%	34.42	43.27
	10	1006	0.00%	-	-	1768	0.00%	81.51	22.82	1814	0.00%	-	338.87
A	Avg		0.00%	177.36	173.42		0.00%	60.57	0.00		0.00%	288.00	103.16
B	1	1006	0.00%	15.03	52.08	1819	0.00%	8.59	3.5	1835	0.00%	23.23	35.56
	2	966	0.00%	93.55	116.12	1848	0.00%	64.6	36.36	1848	0.00%	57.31	33.26
	3	1026	0.00%	13.15	15.46	1834	0.00%	1.79	2.72	1842	0.00%	8.24	40.27
	4	982	0.00%	334.76	315.12	1845	0.00%	83.9	30.31	1850	0.00%	354.32	65.39
	5	1091	0.00%	376.13	286.33	1886	0.53%	101.12	88.5	1897	0.00%	472.52	122.37
	6	954	0.00%	10.25	91.65	1732	0.00%	4.76	2.14	1740	0.00%	9.63	8.15
	7	1034	0.00%	101.9	79.09	1834	0.00%	45.79	38.04	1851	0.00%	1638.57	116.28
	8	1043	0.00%	865.7	151.23	1812	0.00%	17.07	33.63	1812	0.00%	18.09	30.43
	9	1031	0.00%	17.38	137.35	1854	0.00%	215.81	33.2	1854	0.00%	40.08	52.49
	10	1005	-0.10%	-	-	1768	0.00%	49.89	38.26	1814	0.00%	-	556.08
B	Avg		-0.01%	203.09	138.27		0.05%	59.33	0.00		0.00%	291.33	106.03
C	1	1077	7.06%	513.24	259.72	2014	10.72%	180.4	210.34	2014	9.75%	126.37	247.74
	2	968	0.21%	21.13	61.22	1861	0.70%	1604.47	81.65	1861	0.70%	1621.47	80.73
	3	1080	5.26%	13.46	36.99	1848	0.76%	2.41	3.08	1848	0.33%	4.5	2.56
	4	1036	5.50%	68.14	80.43	1911	3.58%	-	291.57	1911	3.30%	-	355.95
	5	1464	34.19%	1717.63	2346.76	2198	17.16%	19.5	183.63	2198	15.87%	21.29	138.42
	6	970	1.68%	17.26	51.63	1756	1.39%	5.61	2.71	1770	1.72%	74.25	95.81
	7	1034	0.00%	82.54	107.31	1861	1.47%	110.7	62.73	1861	0.54%	115.61	83.54
	8	1046	0.29%	212.37	135.45	1812	0.00%	4.33	17.81	1812	0.00%	4.42	19.22
	9	1077	4.46%	948.6	652.17	1930	4.10%	79.43	58.99	1930	4.10%	39.59	56.08
	10	1008	0.20%	-	1812.5	1768	0.00%	5.86	4.94	1814	0.00%	1769.3	229.4
C	Avg		5.88%	399.37	554.42		3.99%	223.63	0.00		3.63%	419.64	130.95
D	1	1082	7.55%	29.59	135.19	1896	4.23%	178.24	136.26	1896	3.32%	201.72	81.01
	2	984	1.86%	23.16	29.04	1905	3.08%	253	33.79	1905	3.08%	285.22	30.02
	3	1046	1.95%	10.77	65.05	1935	5.51%	36.97	119.56	1935	5.05%	31.79	39.17
	4	1155	17.62%	-	664.03	1954	5.91%	-	630.64	1954	5.62%	-	589.41
	5	1137	4.22%	2553.9	581.63	1979	5.49%	100.41	55.76	1979	4.32%	74.97	86.89
	6	954	0.00%	3.48	33.01	1740	0.46%	39.95	4.23	1740	0.00%	13.34	7.73
	7	1060	2.51%	48.3	84.21	1899	3.54%	20.73	83.24	1899	2.59%	18.57	111.19
	8	1043	0.00%	653.97	154.21	1812	0.00%	39.03	11.73	1812	0.00%	47.46	28.93
	9	-	-	-	-	-	-	-	-	-	-	-	-
	10	1012	0.60%	-	1205.99	1847	4.47%	345.26	254.34	1847	1.82%	269.18	241.53
D	Avg		4.03%	474.74	328.04		3.63%	126.70	0.00		2.87%	117.78	135.10

Computational results on SSCFLP instances, N = 100 (cont'd on the next page)

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
E	1	1014	0.80%	14.82	31.35	1864	2.47%	19.08	4.54	1864	1.58%	20.05	5.47
	2	966	0.00%	29	20.61	1862	0.76%	25.79	19.2	1862	0.76%	24.83	13.08
	3	1029	0.29%	4.52	19.64	1842	0.44%	6.05	4.95	1842	0.00%	4.37	5.23
	4	990	0.81%	18.72	114.3	1867	1.19%	117.85	34.87	1867	0.92%	93.79	45.01
	5	1091	0.00%	43.82	127.41	1931	2.93%	38.73	68.86	1931	1.79%	82.88	107.76
	6	960	0.63%	19.58	66.91	1772	2.31%	13.12	27.05	1772	1.84%	22.59	20.44
	7	1056	2.13%	698.57	141.39	1869	1.91%	122.1	31.98	1869	0.97%	43.26	18.26
	8	1049	0.58%	12.6	6.24	1815	0.17%	4.21	3.38	1815	0.17%	4.08	3.97
	9	1036	0.48%	37.84	234.26	1907	2.86%	15.23	33.91	1907	2.86%	13.84	34.25
	10	1005	-0.10%	-	409.08	1840	4.07%	67.42	63.53	1840	1.43%	139.12	93.6
E	Avg		0.56%	97.72	117.12		1.91%	42.96	0.00		1.23%	44.88	34.71
F	1	1020	1.39%	3.05	39.05	1862	2.36%	35.51	5.34	1877	2.29%	17.84	30.62
	2	966	0.00%	32.83	53.94	1851	0.16%	21.54	32.64	1851	0.16%	21.36	34.95
	3	1032	0.58%	3.26	5.68	1839	0.27%	3.97	2.07	1849	0.38%	44.69	11.16
	4	988	0.61%	261.95	236.38	1851	0.33%	85	34.85	1874	1.30%	-	220.17
	5	1109	1.65%	152.23	109.61	1902	1.39%	3.88	1.87	1949	2.74%	108.74	138.02
	6	974	2.10%	17.45	86.05	1764	1.85%	20.81	2	1796	3.22%	17.42	25.82
	7	1036	0.19%	137.07	68.78	1834	0.00%	22.57	19.32	1851	0.00%	364.59	35.68
	8	1043	0.00%	10.53	80.73	1812	0.00%	5.76	3.33	1812	0.00%	3.88	2.73
	9	1036	0.48%	40.01	188.75	1886	1.73%	10.32	3.28	1899	2.43%	27.85	62.37
	10	1009	0.30%	-	1193.61	1803	1.98%	54.57	15.86	1854	2.21%	659.57	212.28
F	Avg		0.73%	73.15	206.26		1.01%	26.39	0.00		1.47%	140.66	77.38
G	1	1027	2.09%	69.35	135.44	1849	1.65%	6.38	3.51	1882	2.56%	21.53	52.14
	2	977	1.14%	233.75	121.98	1848	0.00%	23.22	4.93	1848	0.00%	21.26	3.47
	3	1043	1.66%	13.44	30.43	1867	1.80%	5.64	2.11	1875	1.79%	5.05	2.27
	4	982	0.00%	414.54	284.28	1847	0.11%	51.05	11.57	1869	1.03%	-	134.18
	5	1127	3.30%	520.23	126.18	1916	2.13%	42.98	57.01	1929	1.69%	9.73	125.11
	6	986	3.35%	12.28	110.19	1825	5.37%	9.77	1.78	1874	7.70%	51.44	36.14
	7	1037	0.29%	269.35	146.99	1834	0.00%	40.38	30.93	1901	2.70%	-	769.95
	8	1043	0.00%	445.19	307.44	1813	0.06%	6.8	3.08	1813	0.06%	3.1	20.01
	9	1037	0.58%	108.42	183.79	1863	0.49%	7.96	3.14	1910	3.02%	113.29	72.16
	10	1017	1.09%	-	1307.76	1768	0.00%	76.79	17.07	1814	0.00%	118.87	255.88
G	Avg		1.35%	231.84	275.45		1.16%	27.10	13.51		2.05%	43.03	147.13
Solved instances				61.00	67.00			67.00	63.00			62.00	69.00
Avg. computing time				236.75	256.14			80.95	13.51			192.19	104.92

Table 7.5: Computational results on SSCFLP instances, N = 100

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
A	1	713	0.00%	2.79	6.16	1217	0.00%	0.33	0.28	1242	0.00%	7.19	2.87
	2	740	0.00%	0.9	0.42	1169	0.00%	0.61	0.26	1169	0.00%	0.56	0.25
	3	751	0.00%	1.72	1.39	1141	0.00%	1.42	0.4	1162	0.00%	0.81	0.47
	4	651	0.00%	0.91	0.59	1141	0.00%	1.28	0.44	1141	0.00%	0.67	0.47
	5	664	0.00%	1.07	0.81	1103	0.00%	3.73	0.89	1103	0.00%	0.43	0.52
	6	778	0.00%	0.58	0.67	1240	0.00%	1.43	0.44	1257	0.00%	6.36	9.49
	7	787	0.00%	4.09	17.17	1175	0.00%	1.8	0.67	1213	0.00%	1.74	4.69
	8	820	0.00%	1989.29	202.28	1184	0.00%	1.66	0.31	1248	0.00%	4.61	8.19
	9	715	0.00%	2.16	9.98	1141	0.00%	1.84	0.35	1152	0.00%	1.91	0.8
	10	829	0.00%	9.33	71.52	1159	0.00%	0.73	0.29	1248	0.00%	26.37	45.52
A	Avg		0.00%	201.28	31.10		0.00%	1.48	0.43		0.00%	5.07	7.33
B	1	713	0.00%	5.67	4.89	1217	0.00%	0.32	0.27	1242	0.00%	9.93	3.39
	2	740	0.00%	1	0.36	1169	0.00%	0.27	0.27	1169	0.00%	0.64	0.27
	3	751	0.00%	2.08	1.21	1150	0.79%	1.22	0.58	1162	0.00%	0.78	0.46
	4	651	0.00%	1.12	0.64	1141	0.00%	0.29	0.47	1141	0.00%	0.67	0.47
	5	664	0.00%	0.91	0.76	1103	0.00%	3.1	0.53	1103	0.00%	0.37	0.41
	6	778	0.00%	1.05	0.71	1240	0.00%	1.56	0.45	1257	0.00%	7.41	14
	7	787	0.00%	5.08	24.78	1175	0.00%	1.1	0.57	1213	0.00%	2.37	8.44
	8	821	0.12%	2142.42	203.07	1190	0.51%	0.48	0.47	1248	0.00%	2.91	7.85
	9	715	0.00%	1.77	15.73	1141	0.00%	0.89	0.37	1152	0.00%	1.96	0.7
	10	829	0.00%	9.95	83.41	1183	2.07%	0.95	0.59	1248	0.00%	32.15	47.76
B	Avg		0.01%	217.11	33.56		0.34%	1.02	0.46		0.00%	5.92	8.38
C	1	713	0.00%	1.79	2.05	1258	3.37%	1.13	0.52	1258	1.29%	0.76	0.44
	2	761	2.84%	0.81	0.51	1216	4.02%	1.57	1.77	1216	4.02%	2.26	0.55
	3	751	0.00%	1.99	1.3	1155	1.23%	0.95	0.53	1162	0.00%	0.74	0.46
	4	652	0.15%	0.82	0.5	1141	0.00%	0.5	0.5	1141	0.00%	0.7	0.47
	5	664	0.00%	1.28	0.94	1103	0.00%	2.37	0.51	1103	0.00%	0.34	0.35
	6	799	2.70%	1.02	0.93	1274	2.74%	7.18	13.41	1274	1.35%	3.5	10.85
	7	787	0.00%	8.03	29.17	1175	0.00%	0.71	0.43	1213	0.00%	2.21	4.88
	8	820	0.00%	1702.21	167.71	1195	0.93%	0.7	0.29	1248	0.00%	3.86	9.36
	9	715	0.00%	0.93	0.55	1152	0.96%	1.8	0.61	1152	0.00%	1.57	0.59
	10	835	0.72%	37.26	59.06	1183	2.07%	0.54	0.37	1248	0.00%	19.19	26.98
C	Avg		0.64%	175.61	26.27		1.53%	1.75	1.89		0.67%	3.51	5.49
D	1	744	4.35%	3.7	5.55	1270	4.35%	0.78	0.37	1270	2.25%	0.69	0.4
	2	770	4.05%	2.53	0.88	1209	3.42%	2.53	0.69	1209	3.42%	2.23	0.74
	3	755	0.53%	2.61	1.91	1186	3.94%	2.07	2.01	1186	2.07%	4.1	0.66
	4	651	0.00%	0.97	0.59	1163	1.93%	0.41	0.48	1165	2.10%	1.41	0.44
	5	664	0.00%	0.64	0.8	1103	0.00%	0.92	0.34	1103	0.00%	0.717	0.44
	6	778	0.00%	0.79	0.7	1240	0.00%	1.35	0.46	1257	0.00%	4.7	9.59
	7	845	7.37%	9.55	24.22	1315	11.91%	45.34	18.25	1315	8.41%	45.76	18.81
	8	820	0.00%	71.68	26.44	1285	8.53%	2.66	6.95	1285	2.96%	2.38	6.97
	9	717	0.28%	3.08	21.18	1146	0.44%	0.72	0.37	1165	1.13%	1.83	6.34
	10	848	2.29%	32.91	88.74	1217	5.00%	0.41	0.44	1253	0.40%	9.22	15.71
D	Avg		1.89%	12.85	17.10		3.95%	5.72	3.04		2.27%	7.30	6.01

Computational results on CCLP instances, N = 50 (cont'd on the next page)

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
E	1	714	0.14%	1.87	0.7	1266	4.03%	2.85	3.28	1266	1.93%	3.29	3.26
	2	740	0.00%	0.71	0.34	1170	0.09%	0.65	0.24	1170	0.09%	0.66	0.24
	3	751	0.00%	1.3	0.98	1162	1.84%	0.76	0.36	1162	0.00%	0.85	0.35
	4	652	0.15%	0.72	0.42	1141	0.00%	1.2	0.37	1141	0.00%	1.2	0.37
	5	664	0.00%	1.13	0.85	1103	0.00%	0.73	0.27	1103	0.00%	0.77	0.27
	6	787	1.16%	2.2	0.63	1274	2.74%	3.18	9.31	1274	1.35%	4.08	9.25
	7	789	0.25%	2.33	20.35	1255	6.81%	41.83	17.23	1255	3.46%	24.35	17.25
	8	822	0.24%	667.17	50.25	1266	6.93%	1.46	4.13	1266	1.44%	1.72	4.09
	9	718	0.42%	3.29	10.07	1183	3.68%	2.5	6.16	1183	2.69%	2.45	6.14
	10	829	0.00%	1.81	35.08	1277	10.18%	78.67	29.24	1277	2.32%	59.32	29.21
E	Avg		0.24%	68.25	11.97		3.63%	13.38	7.06		1.33%	9.87	7.04
F	1	713	0.00%	3.92	4.66	1217	0.00%	0.92	0.24	1244	0.16%	3.2	2.9
	2	750	1.35%	1.12	0.84	1181	1.03%	0.91	0.29	1181	1.03%	0.86	0.25
	3	755	0.53%	2.65	2.73	1148	0.61%	1.53	0.35	1177	1.29%	2.43	0.93
	4	651	0.00%	0.94	0.55	1141	0.00%	1	0.4	1141	0.00%	0.62	0.42
	5	666	0.30%	1.08	0.74	1161	5.26%	3.61	1.62	1161	5.26%	0.87	0.53
	6	778	0.00%	1.06	0.66	1248	0.65%	2.06	3.77	1257	0.00%	3.1	6.18
	7	787	0.00%	2.62	21.56	1179	0.34%	1.88	0.73	1232	1.57%	13.23	15.73
	8	820	0.00%	1606.42	87.3	1195	0.93%	0.76	0.28	1248	0.00%	2.97	10.85
	9	715	0.00%	2.41	10.14	1145	0.35%	1.04	0.27	1156	0.35%	1.55	0.52
	10	829	0.00%	5.16	45.19	1188	2.50%	1.81	3.12	1261	1.04%	19.63	28.56
F	Avg		0.22%	162.74	17.44		1.17%	1.55	1.11		1.07%	4.85	6.69
G	1	714	0.14%	2.19	1.85	1217	0.00%	0.71	0.28	1244	0.16%	15.44	2.97
	2	740	0.00%	0.73	0.37	1181	1.03%	0.93	0.23	1181	1.03%	1.38	0.22
	3	751	0.00%	0.81	0.51	1152	0.96%	0.98	0.27	1177	1.29%	0.78	0.38
	4	652	0.15%	1.27	0.55	1141	0.00%	1.15	0.53	1141	0.00%	0.83	0.43
	5	664	0.00%	1.17	0.74	1103	0.00%	0.44	0.29	1103	0.00%	0.602	0.26
	6	778	0.00%	1.02	0.59	1240	0.00%	1.14	0.42	1257	0.00%	4.12	5.58
	7	805	2.29%	7.89	23.76	1197	1.87%	1.24	0.55	1274	5.03%	17.1	17.58
	8	829	1.10%	742.11	99.29	1232	4.05%	0.7	0.48	1267	1.52%	8.97	13.21
	9	715	0.00%	3.91	9.69	1141	0.00%	0.71	0.31	1171	1.65%	0.93	0.48
	10	829	0.00%	3.39	33.45	1183	2.07%	1.36	0.44	1289	3.29%	289.07	39.83
G	Avg		0.37%	76.45	17.08		1.00%	0.94	0.38		1.40%	33.92	8.09
Solved instances				70.00	70.00			70.00	70.00			70.00	70.00
Avg. computing time				130.61	22.07			3.69	2.05			10.06	7.00

Table 7.6: Computational results on CCLP instances, N = 50

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
A	1	1006	0.00%	26.84	79.34	1819	0.00%	8.15	2.89	1835	0.00%	30.07	37.9
	2	966	0.00%	96.04	95.86	1848	0.00%	61.14	22.86	1848	0.00%	78.58	24.36
	3	1026	0.00%	13.88	27.59	1834	0.00%	5.12	2.48	1842	0.00%	19.8	16.23
	4	982	0.00%	234.13	210.62	1845	0.00%	121.26	36.39	1850	0.00%	418.73	88.07
	5	1091	0.00%	385.7	312.55	1876	0.00%	63.01	21.85	1897	0.00%	432.94	115.58
	6	954	0.00%	5.07	92.97	1732	0.00%	10.94	2.11	1740	0.00%	20.47	6.07
	7	1034	0.00%	125.05	112.1	1834	0.00%	60.19	15.77	1851	0.00%	1700.4	111.42
	8	1043	0.00%	801.15	225.63	1812	0.00%	20.07	20.68	1812	0.00%	38.43	27.34
	9	1031	0.00%	16.27	185.24	1854	0.00%	336.79	21.4	1854	0.00%	41.76	35.92
	10	1009	0.00%	-	-	1768	0.00%	193.46	22.68	1814	0.00%	-	563.39
A	Avg		0.00%	189.35	149.10		0.00%	88.01	0.00		0.00%	309.02	102.63
B	1	1006	0.00%	29.83	76.75	1819	0.00%	3.59	3.74	1835	0.00%	17.88	45.4
	2	966	0.00%	147.42	100.78	1848	0.00%	85.45	27.99	1848	0.00%	77.28	22.72
	3	1026	0.00%	17.24	34.53	1834	0.00%	1.8	2.37	1842	0.00%	12.88	12.95
	4	982	0.00%	318.39	302.26	1845	0.00%	83.6	28.99	1850	0.00%	322.68	154.24
	5	1091	0.00%	527.71	215.07	1886	0.53%	132.47	42.22	1897	0.00%	586.07	226.39
	6	954	0.00%	14.31	151.42	1732	0.00%	6.47	2.04	1740	0.00%	11.08	3.89
	7	1034	0.00%	110.62	90.89	1834	0.00%	45.29	35.37	1851	0.00%	2006.64	75.54
	8	1043	0.00%	1102.96	291.21	1812	0.00%	26.07	29.7	1812	0.00%	28.35	30.97
	9	1031	0.00%	17.77	232	1854	0.00%	450.44	39.76	1854	0.00%	31.56	38.95
	10	1005	-0.40%	-	-	1768	0.00%	173.77	33.1	1814	0.00%	-	254.55
B	Avg		-0.04%	254.03	166.10		0.05%	100.90	0.00		0.00%	343.82	86.56
C	1	1077	7.06%	488.07	396.71	2014	10.72%	223.26	165.67	2014	9.75%	197.89	206.83
	2	968	0.21%	37.93	57.88	1861	0.70%	1844.79	93.03	1861	0.70%	1782.91	69.62
	3	1080	5.26%	8.51	47.26	1848	0.76%	1.99	2.74	1848	0.33%	1.85	2.61
	4	1036	5.50%	60.25	61.63	1911	3.96%	-	286.15	1911	3.68%	-	125.93
	5	1464	34.19%	1229.49	2395.68	2198	17.16%	29.42	97.58	2198	15.87%	33.88	114.57
	6	970	1.68%	29.83	61.27	1756	1.39%	4.77	3.82	1770	1.72%	79.99	23.29
	7	1034	0.00%	83.06	145.29	1861	1.47%	116.95	66.91	1861	0.54%	122.32	54.17
	8	1046	0.29%	243.77	131.71	1812	0.00%	3.6	7.31	1812	0.00%	3.5	22.08
	9	1077	4.46%	544.75	503.58	1930	4.10%	149.92	67.29	1930	4.10%	59.19	47.88
	10	1008	-0.10%	-	2487.29	1768	0.00%	9.19	6.6	1814	0.00%	3076.97	195.62
C	Avg		5.85%	302.85	628.83		4.03%	264.88	0.00		3.67%	595.39	86.26
D	1	1082	7.55%	25.13	136.42	1896	4.23%	541.27	61.75	1896	3.32%	707.12	82.34
	2	984	1.86%	34.68	24.46	1905	3.08%	378.81	44.4	1905	3.08%	416.34	41.19
	3	1046	1.95%	5.96	47.97	1935	5.51%	69.47	61.3	1935	5.05%	71.78	40.46
	4	1155	19.25%	-	851.72	1954	5.91%	-	352.02	1954	5.62%	3661	690.02
	5	1137	4.22%	1169.39	1071.69	1979	5.49%	126.98	64.11	1979	4.32%	110.15	66.64
	6	954	0.00%	2.56	69.86	1740	0.46%	21.09	6.56	1740	0.00%	8.96	6.6
	7	1060	2.51%	34.98	77.52	1899	3.54%	15.92	103.92	1899	2.59%	16.49	92.62
	8	1043	0.00%	845.98	437.8	1812	0.00%	34.3	16.22	1812	0.00%	58.2	23.84
	9	-	-	-	-	-	-	-	-	-	-	-	-
	10	1012	0.30%	-	456.6	1847	4.47%	593.93	124.35	1847	1.82%	493.66	135.71
D	Avg		4.18%	302.67	352.67		3.63%	222.72	0.00		2.87%	615.97	131.05

Computational results on CCLP instances, N = 100 (cont'd on the next page)

		CPMP				SS-CFLP				CARD+FIX			
R. Type	Inst.	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time	Avg	cost increase	BP Time	BC Time
E	1	1014	0.80%	23.62	251.21	1864	2.47%	21.16	21.86	1864	1.58%	21.69	21.95
	2	966	0.00%	32.14	23.74	1862	0.76%	57.28	12.95	1862	0.76%	47.48	13.09
	3	1029	0.29%	1.99	5.78	1842	0.44%	2.22	4.27	1842	0.00%	3.01	4.31
	4	990	0.81%	44.7	49.54	1867	1.19%	104.2	44.91	1867	0.92%	76.43	45.13
	5	1091	0.00%	61.7	97.65	1931	2.93%	79.51	75.74	1931	1.79%	118.32	76.25
	6	960	0.63%	48.44	131.38	1772	2.31%	32	16.93	1772	1.84%	38.6	16.92
	7	1056	2.13%	1235.52	143.33	1869	1.91%	152.4	16.6	1869	0.97%	84.82	16.63
	8	1049	0.58%	12.57	46.01	1815	0.17%	3.1	3.11	1815	0.17%	3.92	3.14
	9	1036	0.48%	41.59	234.74	1907	2.86%	14.12	33.79	1907	2.86%	19.01	33.89
	10	1005	-0.40%	3661	403.3	1840	4.07%	104.82	114.49	1840	1.43%	109.16	113.8
E	Avg		0.53%	516.33	138.67		1.91%	57.08	0.00		1.23%	52.24	34.51
F	1	1020	1.39%	2.59	34.27	1862	2.36%	56.29	3.94	1877	2.29%	22.18	38.03
	2	966	0.00%	64.55	56.57	1851	0.16%	69.81	24.15	1851	0.16%	22.87	22.43
	3	1032	0.58%	1.92	6.65	1839	0.27%	5.88	1.93	1849	0.38%	66.74	10.51
	4	988	0.61%	958.77	358.1	1851	0.33%	112.76	27.49	1874	1.30%	3661	273.69
	5	1109	1.65%	183.12	158.03	1902	1.39%	4.78	1.73	1949	2.74%	144.38	141.36
	6	974	2.10%	11.05	166.72	1764	1.85%	33.32	1.87	1796	3.22%	21.24	67.55
	7	1036	0.19%	133.31	66.98	1834	0.00%	35.01	20.13	1851	0.00%	606.29	78.48
	8	1043	0.00%	11.76	75.27	1812	0.00%	6.4	3.11	1812	0.00%	3.29	3.46
	9	1036	0.48%	18.02	178.43	1886	1.73%	28.03	4.31	1899	2.43%	23.82	62.28
	10	1009	0.00%	-	699.05	1803	1.98%	222.18	26.82	1854	2.21%	197.66	408.88
F	Avg		0.70%	153.90	180.01		1.01%	57.45	0.00		1.47%	476.95	110.67
G	1	1027	2.09%	93.48	109.17	1849	1.65%	7.21	3.51	1882	2.56%	11.83	61.79
	2	977	1.14%	267.2	167.44	1848	0.00%	51.77	8.04	1848	0.00%	31.49	7.07
	3	1043	1.66%	41.32	55.09	1867	1.80%	4.77	2.18	1875	1.79%	3.55	2.11
	4	982	0.00%	662.23	227.18	1847	0.11%	177.36	11.77	1869	1.08%	-	72.14
	5	1127	3.30%	500.26	124	1916	2.13%	59.01	40.03	1929	1.69%	8.32	46.28
	6	986	3.35%	18.83	88.46	1825	5.37%	31.98	1.53	1874	7.70%	42.17	28.24
	7	1037	0.29%	328.74	195.26	1834	0.00%	88.7	15.34	1901	2.70%	-	379.74
	8	1043	0.00%	1603.61	152.28	1813	0.06%	6.62	2.47	1813	0.06%	2.55	17.42
	9	1037	0.58%	88.21	142.41	1863	0.49%	6.44	2.43	1910	3.02%	179.88	87.22
	10	1017	0.79%	3661	1052.6	1768	0.00%	94.35	10.42	1814	0.00%	177.12	46.38
G	Avg		1.32%	726.49	231.39		1.16%	52.82	9.77		2.06%	57.11	74.84
Solved instances				63.00	67.00			67.00	63.00			64.00	69.00
Avg. computing time				349.37	263.82			120.55	9.77			350.07	89.50

Table 7.7: Computational results on CCLP instances, N = 100

Chapter 8

A final note

As a final assessment, we try to gather the results presented in each chapter. This aims at providing a guideline for devising effective algorithms for partitioning problems.

8.1 Using dual information for bounding and variable fixing

A first key issue in decomposition methods is that of finding a technique with good convergence properties for computing the dual bound. The design of such method is again a combination of different aspects.

The first choice to be made is which classes of constraints should be put in the master problem and which should be treated in the subproblems. As recalled in Chapter 1, the space described by the constraints in the pricing problem is convexified, so it is tempting to treat in that way as much constraints as possible. If the pricing problem has the integrality property, there is no improvement in the dual bound with respect to the LP relaxation bound. On the other hand, the pricing problem has to be solved iteratively, and so it should be still tractable.

The design of an effective branch-and-price algorithm heavily relies on identifying, or devising, effective methods to solve non trivial pricing problems. The partitioning problems described in the introduction can be decomposed in such a way that the pricing problem is either a knapsack problem (KP) [74] [55] or an extension of it. The KP is a typical example of a $\mathcal{NP}$-hard combinatorial optimization problem well solved in practice [72] [89]. It is trivial to say that devising a fast procedure for the pricing, and restricting the generation of columns to the optimal ones only is far better than resorting to heuristics and relaxations, also from an experimental point of view. However, it is untrivial to observe that even a few additional constraints may complicate the elegant structure of the algorithms for the KP. In our experience, the procedure for pricing has still to be tailored to the particular problem at hand. For instance, the pricing routine introduced for the packing algorithms of Chapters 3 and 4 actually improved the performance of the method. It is even more interesting to note that it was experimentally useful to search for *the* best column instead of considering *a set* of good columns. This is easily explained with the observation that a set of good columns is useful only if the columns encode a good variety of solutions.

A second factor impacting on the convergence of the method is the presence of stability problems during the iterative solution of the restricted master linear programs (RMPs). The problem of stability received much attention recently [11]. As observed in [92], due to the high ratio between the number of variables and the number of constraints, the optimal solution of a RMP is often degenerate. This corresponds to poor stability in the dual solution: the dual variables assume extreme values, the pricing routine identifies columns coding vertices far away from the MP optimum, and the poor quality of these columns causes the solution of the RMP to be almost the same, for several iterations. An idea on the stability of a dual solution can be obtained by looking at the values of the Lagrangean relaxation corresponding to each dual solution. Instability appears as subsequent relaxed solutions assuming values in a wide range.

Three families of methods have been devised to overcome this problem. The first one is that of *box* methods [33]. A specialization of this technique for partitioning problems is presented in [98]. The main idea is to bound the value of the dual variables, or penalize their deviation from a central value, adding suitable constraints in the dual problem. Each constraint in the dual problem corresponds to a column in the primal, hence these have no effect on the pricing routine. However, an additional designing effort is needed, since several parameters should be tuned in order to make these methods work effectively.

In our experience, the second way of obtaining a smooth improvement of the dual is to solve the pricing problem using a dual solution that is an *interior point* of the optimal dual polyhedron [92]. In fact, it is a common practice to relax the set partitioning formulation into a set covering formulation (see Chapters 3, 4, 5 and 6). Once a RMP is optimized with the simplex algorithm, the optimal solution corresponds to a vertex of the dual polyhedron. Instead, an inner point of the space of the optimal dual solutions would yield the generation of more balanced columns. Therefore, after the computation of each linear program, the structure of the optimal solution can be exploited by enforcing the fulfilment of the complementary slackness conditions. This can be done by changing the sense of the constraints and the right-hand-side terms in the model. In this way the space of valid dual solutions is restricted to the space of optimal dual solutions. The resulting problem is optimized several times with random objective functions, in order to obtain solutions corresponding to different vertices of the optimal dual polyhedron. Then, an inner point is found averaging on the coordinates of the dual solutions identified in this way.

A third method is resorting to Lagrangean/Surrogate relaxation [96]. Once a RMP is solved, the corresponding dual solution is used to compute a Lagrangean relaxation bound (see, for instance, Chapter 6). Then, the value of all the dual variables is scaled by a parameter t, and a new Lagrangean dual bound is computed. The value of the parameter t that gives the best Lagrangean bound can be found by dicotomic search. Finally, the scaled dual vector is used for pricing.

The second method is well-suited for applications in which the pricing problem is very hard to solve, as it shifts all the computing effort for stabilization to solving a sequence of RMP instances.The third one works well for applications in which the pricing problem is easy, since it requires to run the pricing procedure for each choice of the parameter. The first one is the most versatile, but has the drawback of requiring a fine parameter tuning.

As in the third method, we improved the convergence properties and avoided instability exploiting the equivalence between Dantzig-Wolfe decomposition and Lagrangean relaxation. In particular, we devised a multiple pricing routine, that instead of computing the best columns corresponding to the optimal dual variables only, further explores the surrounding dual space with a standard subgradient technique. In this way vector of dual variables can be found, that yields better Lagrangean relaxation values. Whenever such a vector is found, the columns corresponding to the Lagrangean relaxed solution are inserted in the RMP. The schema in Figure 8.1(a) represents the generation of columns with a standard pricing method, when the space corresponding to the variables of the original formulation is considered: each dot represents a column, that is an integer point in the space of the original variables. Initially, only a subset of the feasible points is known (those represented in black in the figure), corresponding to columns inserted in the RMP. After the optimization of the RMP, extreme vertices of the convexified constraints region are generated and added to the RMP (represented in light gray in the figure). In Figure 8.1(b) we represent the effect of coupling the traditional pricing with a subgradient-like search for better dual solutions. During this search, vertices far from the current optimal one can be encountered, that may nevertheless represent basic columns of the optimal solution (marked with the large circle in the figure).

This technique was used in the algorithms presented in Chapters 5, 6 and 7. It always showed to be experimentally useful. In order to obtain a good updating step in the subgradient procedure, the structure of the best column for each class must be known at each iteration. The more accurate this information is, the more effective the updating step is. When the structure of the optimal column for a

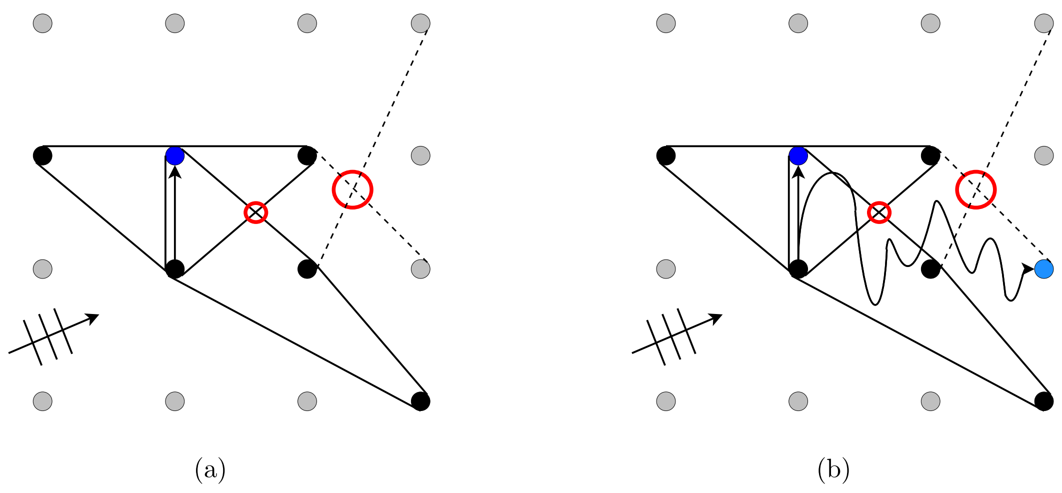


Figure 8.1: Generation of new columns

class is not known, we found it useful to look at a relaxed subproblem solution, instead of computing a heuristic one. In the algorithm for the OOEBPP, described in Chapter 4, the fractional solution of the subproblems is used in place of the integer one. Experimental results show the effectiveness of this technique.

Furthermore, each time the solution of a Lagrangean Relaxed is computed, variable fixing procedures can be activated. In most cases, these reduction tests can be done with a little computational effort. Once again, the application of reduction procedures exploiting exact information on the subproblem solution is described for location problems in Chapter 7, while procedures that use an approximation are devised for packing problems, as described in Chapters 3 and 4. Especially in the case of location problems, these showed be essential in reducing the size of the problem.

8.2 Using the fractional solutions for branching and heuristics

8.2.1 Branching rules

Branching is a key issue in any implicit enumeration algorithm. When a column generation method is applied, an additional designing effort is required, since many variables of the problem are not considered. A discussion about common pitfalls in this context is presented, for instance, in [8] and [51]. When dealing with decomposition methods, instead of branching on the variables of the MP, it is better to rebuild a fractional solution of the original formulation and to branch on the original variables. When considering partitioning problems, a (fractional) solution for the original formulation (1.4) – (1.6), described by a set of vectors x^j can be found by looking at a (fractional) solution of the master problem (1.9) – (1.10) as follows:

$$x^j = \sum_{k \in K^j} \bar{x}^j_k z^j_k.$$

Each component x^j_i represents how much the element i is assigned to class j in the fractional solution. We used a similar technique in all the algorithms presented in this thesis.

The most effective way of branching must still be tailored to the specific application. However, the following guidelines yielded good results in our contexts: if there are variables that represent the opening of a class, like setup variables in location problems, it is better to fix the value of this variables through a binary branching rule. This method was applied in the packing algorithms presented in Chapters 3 and 4, and showed to be the best approach for some of the models presented in Chapter 7. In both cases, in fact, the fixing of these variables suffices to either significantly improve the incumbent primal solution or fathom the node.

If no such set of variables is present in the model, or whenever the relaxed solution is still fractional after the exploration of the first stage branching tree, binary branching on partitioning constraints is the most effective strategy: the element i whose component is non-zero in the maximum number of x^j vectors is selected, and the set of classes is partitioned in two subsets, forbidding in each branch the assignment of i to the classes in one of the two subsets. For instance, we adopted this approach in the algorithm for the multilevel assignment problem of Chapter 5, and for the capacitated p-median problem in Chapter 6. In the first case, a simple variation in which a third branch is considered and explored in a depth-first fashion allowed a substantial improvement in the computing time.

8.2.2 Finding feasible solutions

In primal heuristics, like in branching rules, it is better to run rounding procedures on the fractional solution of the original formulation instead of trying to search for integer solutions of the MP. The framework described by Martello and Toth for the GAP seems to be the most appropriate in this context [73]. Let $J(i) \subseteq J$ be the set of classes to which element i can be assigned, without violating the corresponding block of constraints (1.2). Let, for each element i, $j'(i) \in \text{argmax}_{j \in J(i)} x^j_i$ be the class corresponding to the highest fractional assignment. Each $j'(i)$ can be considered the 'most desirable' class in which i can be inserted. In a similar way, we search for the second 'most desirable' class: $j''(i) \in \operatorname{argmax}_{j \in J(i) \setminus \{j'(i)\}} x_i^j$. Then, we select the element i with maximum *regret value* $x_i^{j'(i)} - x_i^{j''(i)}$ and we assign it to $j'(i)$. This implies a resource consumption on the corresponding class, hence some $J(i)$ set should be updated. The computation of the regret values and the assignment of elements to classes is iterated until either a feasible partitioning is found, or some $J(i)$ set is empty. In the second case, the heuristic fails in finding a feasible solution.

We experimentally observed that in many cases, it is easy for this heuristic to identify feasible solution, and coupling this technique with standard neighborhood search yields tight primal bounds already at the root node. In fact, this kind of techniques were embedded in the algorithms of chapters 5, 6 and 7. On the opposite, the absence of assignment costs made this heuristic blind in packing problems like the ones presented in chapters 3 and 4. Furthermore, the RMP solutions produced during the optimization of the OOEBPP (Chapter 4) were highly fractional, and provided misleading values for the heuristic. Even thought, in these cases, randomized heuristics run in a preprocessing step suffice to obtain tight primal bounds, effectively exploiting fractional solutions to identify upper bounds is still an open problem.

8.3 Research directions

As outlined in the preface, the main objective of this work was to provide a detailed insight of branch-and-price algorithms, by highlight both potential and limits of this approach. We considered the class of partitioning problems, whose models are general enough to describe many real-world scenarios while yielding tractable formulations. Indeed, with similar branch-and-price frameworks we were able to tackle a good variety of them.

The strive is now in identifying techniques to automatically analyze a model, choose a suitable decomposition schema and solve the problem with column generation algorithms. This would definitely unfold the modeling and computational potential of branch-and-price for the realization of a "next-generation" of general purpose optimization tools.

Bibliography

[1] CPLEX 8.1. *ILOG: Homepage http://www.ilog.com*, Last access, 15/09/2005.

[2] K. Aardal. Capacitated facility location: Separation algorithms and computational experience. *Mathematical Programming*, 81:149–175, 1998.

[3] P. Avella, A. Sassano, and I. Vasilév. Computational study of large scale p-median problems. *Optimization Online*, 625, 2003. Optimization Online site : www.optimization-online.org.

[4] L. Bahiense, N. Maculan, and C.A. Sagastizábal. The volume algorithm revisited. relation with bundle methods. *Mathematical Programming*, 94(1):41–69, 2002.

[5] E. Balas and E. Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28:1130–1154, 1980.

[6] R. Baldacci, E. Hadjiconstantinou, V. Maniezzo, and A. Mingozzi. A new method for solving capacitated location problems based on a set partitioning approach. *Computers and Operations Research*, 29:365–386, 2002.

[7] F. Barahona and R. Anbil. The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming A*, 87, 2000.

[8] C. Barnhart, E. L. Johnson, G.L. Nemhauser, M. W.P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 46:316–329, 1998.

[9] J.E. Beasley. A note on solving large p-median problems. *European Journal of Operational Research*, 21:270–273, 1985.

[10] C. Beltran, C. Tadonki, and J. Vial. Solving the p-median problem with a semi-lagrangian relaxation. Technical report, University of Geneva, 2004.

[11] H. Ben Amor, J. Desrosiers, and A. Frangioni. Stabilization in column generation. Technical report, 2004.

[12] J.O. Berkey and P. Y. Wang. Two-dimensional finite bin-packing algorithms. *Journal of the Operational Research Society*, 38:423–429, 1987.

[13] A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-price algorithm for the bi-dimensional level strip packing problem. Talk at AIRO-Winter '05, Cortina d'Ampezzo (Italy), January 2005.

[14] A. Ceselli. Algoritmi branch and bound e branch and price per il problema delle p-mediane con capacità. Master's thesis, Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano, Crema, Italy, 2002. In Italian.

[15] A. Ceselli. Two exact algorithms for the capacitated p-median problem. *4OR*, 1(4):319–340, 2003.

[16] A. Ceselli, F. Liberatore, and G. Righini. A computational evaluation of a general branch-and-price framework for capacitated network location problems. Technical report, DTI – Univeristà di Milano, 2005.

[17] A. Ceselli and G. Righini. A branch-and-price algorithm for the multilevel generalized assignment problem. Technical report, DTI – Università di Milano, 2004.

[18] A. Ceselli and G. Righini. A branch-and-price algorithm for the capacitated p-median problem. *Networks*, 45(3):125 – 142, 2005.

[19] A. Ceselli and G. Righini. An optimization algorithm for a penalized knapsack problem. *Operations Research Letters*, in press, available online, 2005.

[20] A. Ceselli and G. Righini. An optimization algorithm for the ordered open-end bin-packing problem. Talk presented at ECCO XVIII, Minsk, May 2005.

[21] N. Christofides and J.E. Beasley. A tree search algorithm for the p-median problem. *European Journal of Operational Research*, 10:196–204, 1981.

[22] N. Christofides and J.E. Beasley. Extensions to a lagrangean relaxation approach for the capacitated warehouse location problem. *European Journal of Operational Research*, 12:19–28, 1983.

[23] E.G. Coffman Jr., M. R. Garey, and D. S. Johnson. *Approximation algorithms for bin packing: a survey.* PWS Publishing Company, Boston, U.S.A., 1996.

[24] G. Cornueojols, M.L. Fisher, and G.L. Nemhauser. Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, 1977.

[25] J. Current, M. Daskin, and D. Schilling. *Discrete Network Location Models*, pages 81–118. Springer Verlag, Berlin, 2002.

[26] G. Desaulniers, J. Desrosiers, and M.M. Solomon, editors. *Column Generation.* Springer, 2005.

[27] J. Desrosiers and M.E. Lübbecke. Selected topics in column generation. Technical report, HEC Montreal, 2002. to appear in Operations Research.

[28] J.A. Diaz and E. Fernández. A branch-and-price algorithm for the single source capacitated plant location problem. *Journal of the Operational Research Society*, 53:728–740, 2002.

[29] J.A. Diaz and E. Fernández. Hybrid scatter search and path relinking for the capacitated p-median problem. *European journal of Operational Research*, In press.

[30] H. Dickhoff, G. Scheithauer, and J. Terno. *Cutting and packing*, pages 393–413. Wiley, New York, 1997.

[31] J.J. Dongarra. Performance of various computers using standard linear equations software. *University of Tennessee*, 2005. Technical Report, available at http://www.netlib.org/benchmark/performance.ps.

[32] K. Dowsland. Some experiments with simulated annealing techniques for packing problems. *European Journal of Opeerations Research*, 68:389–399, 1993.

[33] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized column generation. *Discrete Mathematics*, 194:229–237, 1999.

[34] S. P. Fekete and J. Schepers. On higher-dimensional packing iii: Exact algorithms. Technical Report 97–290, 1997.

[35] S. P. Fekete and J. Schepers. A combinatorial characterization of higher-dimensional orthogonal packing. *Mathematics of Operations Research*, 29:353–368, 2004.

[36] S. P. Fekete and J. Schepers. A general framework for bounds for higher-dimensional orthogonal packing problems. *Mathematical Methods of Operations Research*, 60(2):311–329, 2004.

[37] A. Frangioni. About lagrangian methods in integer optimization. *Annals of Operations Research*, 139:163 – 193, 2005.

[38] R. Freling, H.E. Romeijn, D.R. Morales, and A.P.M. Wagelmans. A branch-and-price algorithm for the multiperiod single-sourcing problem. *Operations Research*, 51(6):922–939, 2003.

[39] A.P. French and J.M. Wilson. Heuristic solution methods for the multilevel generalized assignment problem. *Journal of Heuristics*, 8:143–153, 2002.

[40] R.D. Galvão. A dual-bounded algorithm for the p-median problem. *Operations Research*, 28(5), 1979.

[41] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness.* W.H. Freeman, New York, 1979.

[42] A.M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical programming studies*, 2:82–114, 1974.

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

[44] F. Glover, J. Hultz, and D. Klingman. Improved computer-based planning techniques. part ii. *Interfaces*, 9(4):12–20, 1979.

[45] E. Gourdin, M. Labbé, and H. Yaman. *Telecommunication in location*, pages 275–305. Springer, Berlin, 2003.

[46] P. Hanjoul and D. Peeters. A comparison of two dual-based procedures for solving the p-median problem. *European Journal of Operational Research*, 20:387–396, 1985.

[47] M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

[48] C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[49] K. Holmberg, M. Rönnqvist, and D. Yuan. An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, 113:544–559, 1999.

[50] S. Jacobs. On genetic algorithms for the packing of polygons. *European Journal of Operations Research*, 88:165–181, 1996.

[51] E.L. Johnson, G.L. Nemhauser, and M.W.P. Savelsbergh. Progress in linear programming based branch-and-bound algorithms: An exposition. *INFORMS Journal on Computing*, 12, 2000.

[52] K. Jörnsten and M. Näsberg. A new lagrangean relaxation approach to the generalized assignment problem. *European journal of Operational Research*, 27:313–323, 1986.

[53] O. Kariv and S.L. Hakimi. Reducibility among combinatorial problems. *SIAM Journal of Applied Mathematics*, 37:539–560, 1979.

[54] R.M. Karp. *Reducibility among combinatorial problems*, pages 85–103. Plumunu Press, New York, 1972.

[55] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer Verlag, 2005.

[56] C. Kenyon and E. Rémila. A near-optimal solution to a two-dimensional cutting stock problem. *Mathematics of Operations Research*, 25:645–656, 2000.

[57] GNU Linear Programming Kit. *Homepage http://www.gnu.org/software/glpk/*, Last accessed 15/09/2005.

[58] J.G. Klincewicz and H. Luss. A lagrangean relaxation heuristic for capacitated facility location with single-source constraints. *Journal of the Operational Research Society*, 37(5):495–500, 1986.

[59] A. Klose and A. Drexl. Facility location models for distribution system design. *European Journal of Operational Research*, 2004.

[60] A. Klose and S. Görtz. *An exact column generation approach to the capacitated facility location problem*, volume 544 of *Lecture Notes in Economics and Mathematical Systems.* Springer, Berlin, 2004.

[61] M. Labbé, D. Peeters, and J.F. Thisse. *Location on Networks, in Network Routing*, volume 8. Elsevier Science B.V., 1995.

[62] M. Labbé and H. Yaman. A note on the projection of polyhedra. *Optimization Online*, 776, 2003. Optimization Online site : www.optimization-online.org.

[63] M. Labbé and H. Yaman. Polyhedral analysis for concentrator location problem. *Optimization Online*, 694, 2003. Optimization Online site : www.optimization-online.org.

[64] M. Laguna, J.P. Kelly, J.L. Gonzalez-Velarde, and F. Glover. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82:176–189, 1995.

[65] A.H. Land and A.G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[66] J.Y.T. Leung, M. Dror, and G. H. Young. A note on an open-end bin packing problem. *Journal of Scheduling*, 4:201–207, 2001.

[67] A. Lodi, S. Martello, and M. Monaci. Two-dimensional packing problems: a survey. *European Journal of Operational Research*, 141:241–252, 2002.

[68] A. Lodi, S. Martello, and D. Vigo. Models and bounds for two dimensional packing problems. *Journal of Combinatorial Optimization*, 8:363 – 379, 2004.

[69] L. Lorena and E. Senne. A column generation approach to capacitated p-median problems. *Computers and Operations Research*, 31(6):863–876, 2004.

[70] V. Maniezzo, A. Mingozzi, and R. Baldacci. A bionomic approach to the capacitated p-median problem. *Journal of Heuristics*, 4:263–280, 1998.

[71] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS journal on computing*, 15:310–319, 2003.

[72] S. Martello, D. Pisinger, and P.Toth. Dynamic programming and strong bounds for the 0–1 knapsack problem. *Management Science*, 45(3):414–424, 1999.

[73] S. Martello and P. Toth. *An algorithm for the generalized assignment problem*, pages 589–603. 1981.

[74] S. Martello and P. Toth. *Knapsack problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

[75] S. Martello and P. Toth. *Knapsack problems: Algorithms and Computer Implementations*. Wiley, New York, 1990.

[76] R.K. Martin. *Large scale linear and integer optimization.* Kluwer academic, 1998.

[77] R.K. Martin. *Large scale linear and integer optimization.* Kluwer, 1999.

[78] J.M. Mulvey and P. Beck. Solving capacitated clustering problems. *European Journal of Operational Research*, 18:339–348, 1984.

[79] A.T. Murray and R.A. Gerrard. Capacitated service and regional constraints in location-allocation modeling. *Location Science*, 5(2):103–118, 1997.

[80] S.C Narula, U.I. Ogbu, and H.M. Samuelsson. An algorithm for the p-median problem. *Operations Research*, 25(4), 1977.

[81] R.M. Nauss. Solving the generalized assignment problem: An optimizing and heuristic approach. *INFORMS Journal on Computing*, 15(3):249–266, 2003.

[82] A.W. Neebe and M.R. Rao. An algorithm for the fixed charge assigning users to sources problem. *Journal of the Operational Research Society*, 34(11):1107–1113, 1983.

[83] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization.* Wiley - Interscience, 1988.

[84] I.H. Osman and N. Christofides. Capacitated clustering problems by hybrid simulated annealing and tabu search. *International Transactions in Operational Research*, 13:317–336, 1994.

[85] M.A. Osorio and M. Laguna. Logic cuts for multilevel generalized assignment problems. *European Journal of Operational Research*, 151:238–246, 2003.

[86] H. Pirkul. Efficient algorithms for the capacitated concentrator location problem. *Computers and Operations Research*, 14(3):197–208, 1987.

[87] D. Pisinger. A minimal algorithm for the multiple–choice knapsack problem. *European Journal of Operational Research*, 83(2):392–410, 1995.

[88] D. Pisinger. A minimal algorithm for the 0–1 knapsack problem. *Operations Research*, 45:758–767, 1997.

[89] D. Pisinger. Where are the hard knapsack problems? *Computers and Operations Research*, 32:2271–2284, 2005.

[90] K.E. Rosing. Towards the solutions of the (generalised) multi-weber problem. *Environment and Planning, Series B*, 18:347–360, 1991.

[91] G.T. Ross and R.M. Soland. Modeling facility location problems as generalized assignment problems. *Management Science*, 24(3), 1977.

[92] L.M. Rousseau, M. Gendreau, and D. Feillet. Interior point stabilization for column generation. Technical report, Laboratoire Informatique d'Avignon, 2003. submitted to Operations Research Letters.

[93] T.J. Van Roy and D. Erlenkotter. Dual-based procedure for dynamic facility location. *Management Science*, 28(10), 1982.

[94] T.J. Van Roy and D. Erlenkotter. A cross decomposition algorithm for capacitated facility location. *Operations Research*, 34:145–163, 1986.

[95] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6), 1997.

[96] E.L.F. Senne and L.A.N. Lorena. Stabilizing column generation using lagrangean/surrogate relaxation: an application to p-median location problems. In *Proceedings of the EURO 2001 conference*, Erasmus University Rotterdam, July 2001.

[97] E.L.F. Senne, L.A.N. Lorena, and M.A. Pereira. A branch-and-price approach to p-median location problems. *Computers and Operations Rsearch*, 32(6):1655–1664, 2005.

[98] M. Sigurd. Stabilizing column generation. In *International Symposium on Mathematical Programming*, Copenhagen, August 2003.

[99] S.S. Syam. A model for the capacitated p-facility location problem in global environments. *Computers Ops Res.*, 24(11):1005–1016, 1997.

[100] F. Vanderbeck. *Decomposition and Column Generation for Integer Programming.* PhD thesis, Université Catholique de Louvain, Louvain, Belgique, 1994.

[101] Xpress-Optimizer. *Dash Optimization: Homepage http://www.dashoptimization.com*, Last accessed 15/09/2005.

[102] M. Yagiura, T. Yamaguchi, and T. Ibaraki. A variable depth search algorithm with branching search for the generalized assignment problem. *Optimization methods and software*, 10:419–441, 1998.

[103] J. Yang and J. Y.T. Leung. The ordered open-end bin-packing problem. *Operations Research*, 51:759–770, 2003.

List of Figures

List of Tables