Université Catholique de Louvain

Faculté des Sciences Appliquées

Unité de Gestion Industrielle

Decomposition and Column Generation for Integer Programs

Thèse présentée en vue de l'obtention du Grade de

Docteur en Sciences Appliquées

par

François Vanderbeck

Promoteur: Professeur **Laurence A. Wolsey**

Louvain-La-Neuve, Septembre 1994.

Decomposition and Column Generation for Integer Programs

by

François Vanderbeck

Abstract

The decomposition of an integer program can provide a reformulation whose linear programming relaxation yields a tight bound. This in turn may allow one to solve difficult integer programs to optimality. However, dealing with an integer program that has a large (implicit) number of columns requires the integration of a column generation procedure in the exact solution algorithm. The standard integer programming techniques must be adapted to become compatible with column generation.

We implement an algorithm combining column generation and branch-and-bound. We propose a branching scheme that offers a unified view of branching rules previously used in a column generation framework and extends the class of problems that can be dealt with. We also derive conditions for early termination of the column generation procedure. We test the algorithm on three types of problems: a telecommunication traffic assignment problem, a graph node clustering problem and a single-machine multi-item lot-sizing problem. We discuss a few algorithmic choices that have a significant influence on the performance of the algorithm. Our computational results highlight the advantages and the limitations of this approach.

Jury: Prof. Guy de Ghellinck (UCL)
Prof. Etienne Loute (Facultés Universitaires St. Louis)
Prof. Yves Pochet (UCL)
Prof. Martin W.P. Savelsbergh (Georgia Institute of Technology)
Prof. **Laurence A. Wolsey** (UCL)

Acknowledgments

I would like to express my special thanks to my supervisor, Laurence Wolsey, who has encouraged me to start this Ph. D. and has supported me ever since. He was always there when I needed him. It was a real pleasure to learn to do research drawing on his large expertise in the field.

I am also very grateful for the constructive comments I have received from the members of the jury: Guy de Ghellinck, Etienne Loute, Yves Pochet, and Martin Savelsbergh. I also thank Jonathan Bard for his useful remarks.

I also would like to take this opportunity to thank the people of the Center for Operations Research and Econometrics (CORE): its staff, my fellow doctoral students, and the professors, for the good company and for the help and advice.

I gratefully acknowledge a doctoral fellowship provided by the Intercollegiate Center for Management science (I.C.M., Brussels).

Contents

1 Introduction

Many practical optimization problems can be formulated as set partitioning problems: given a ground set of elements and a characterization of feasible subsets and their costs, one searches for a selection of feasible subsets that forms a minimum cost partition of the ground set. More generally, one might wish to find a minimum cost selection of subsets that includes each ground set element (at least/at most) a specified number of times. Such applications arise in the area of operational problems like the distributions of goods, but also in planning problems such as facility location problems and assignment problems, and design problems such as communication and transportation network design problems.

To model these optimization problems one can either use a formulation containing a set of constraints that define feasible subsets implicitly, or one can use a formulation that explicitly enumerates all feasible subsets. In many applications, the latter formulation choice improves one's chances of solving the problem to optimality. However, it raises specific difficulties due to the typically large number of feasible subsets one has to deal with. For instance, suppose that the ground set consists of a set of 100 customers and that a feasible subset is a subset of customers to whom goods can be delivered by a single truck. Assuming a truck can fit goods for at most 10 customers, there are on the order of 10^{12} possible subsets.

Our goal is to study the theoretical and practical problems encountered in tackling such large formulations. Building on previous results, we develop an exact solution algorithm for a class of set-partitioning-like problems. Using a partial representation of the large formulation, we generate feasible subsets if and when needed, a technique known as column generation. In particular, we consider a telecommunication traffic assignment problem for which this approach is one of the only practical solution method. We address the issue of the robustness of the methodology across applications by looking at two other problems: a graph partitioning problem and a multi-item single-machine lot-sizing problem.

In the rest of this introduction, we first present the subject of this thesis in more detail. In an informal discussion, we place our topic in its context and motivate its interest. The reader who is familiar with the field of integer programming may want to skip these explanations (i.e. Section 1.1). Next, we give some theoretical background about Dantzig-Wolfe decomposition and column generation, we formalize the decomposition and reformulation of integer programs, and we compare different

relaxations of integer programs. Then we give an overview of the literature in the field. Finally, we discuss the content of this thesis and give a Chapter outline.

1.1 Scope and Motivation

A rich variety of optimization problems can be formulated as maximizing or minimizing a linear function of many variables subject to inequality and equality constraints and integrality restrictions on some or all of the variables. Such models are referred to as (mixed-)*integer programs*.

The successful solution of integer programs is based on solving easier problems, called *relaxations*, to obtain bounds approximating the optimal value. These bounds are used to carry out an implicit enumeration of all feasible solutions. The technique is called *branch-and-bound*.

The quality of the approximations (bounds) is critical for the efficiency of the method. Conventional branch-and-bound uses a linear programming (LP) relaxation of the integer programming formulation to provide bounds. Solving the linear program resulting from the relaxation of all the integrality constraints of a given problem formulation is easy, using say the simplex algorithm; but the resulting approximation is often poor.

In an effort to provide better approximations, other types of relaxation have been used. The *Lagrangean approach* consists of relaxing some of the constraints of a given formulation of the problem as opposed to all integrality constraints. The *polyhedral approach* consists of improving the LP relaxation by adding inequalities to strengthen the formulation.

Another technique that generally induces good approximations is to consider an alternate problem formulation that is more extensive (i.e. that contains a large number of variables). In this thesis we investigate an exact integer programming solution method that relies on the quality of the approximation provided by the extensive formulation.

A problem reformulation with many variables

Let us illustrate on an example what we mean by an extensive reformulation. Given a finite set of items $i \in I = \{1, \ldots, m\}$, their sizes $s_i \geq 0$, and a bin capacity C, one

wishes to place items in bins so that the sum of the size of the items in a bin does not exceed the bin capacity. The problem is to minimize the number of bins needed to pack all items. It is known as the *bin packing problem*. To derive a compact integer programming formulation, we define x_i^k to be 1 if item i is placed in bin k and zero otherwise, and we define y^k to be 1 if bin k is used and zero otherwise. Then a formulation of the problem is:

$$\begin{array}{rcll} \min & \displaystyle\sum_{k=1}^{m} y^k & & \\ \text{s.t.} & & & \\ \displaystyle\sum_{k=1}^{m} x_i^k & = & 1 & \forall\, i = 1, \ldots, m \quad (1) \\ \displaystyle\sum_{i=1}^{m} s_i\, x_i^k & \leq & C\, y^k & \forall\, k = 1, \ldots, m \quad (2) \\ x_i^k, y^k & \in & \{0,1\} & \forall\, i,\ k. \end{array}$$

This formulation involves $m^2 + m$ variables and $2m$ constraints.

Another point of view is to consider all possible ways to place items into a bin. For instance, when $m = 4$, $s = (1, 2, 3, 4)$, and $C = 5$, there are 8 feasible ways to assign items to a bin. In the matrix below, each column represents the incidence vector of a feasible assignment of items to a bin. Letting λ_q be 1 if the assignment q is selected in the solution and zero otherwise, an extensive formulation is:

$$\begin{array}{rcl} \min & \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 + \lambda_5 + \lambda_6 + \lambda_7 + \lambda_8 & \\ s.t. & \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ \lambda_4 \\ \lambda_5 \\ \lambda_6 \\ \lambda_7 \\ \lambda_8 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\ & (\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6, \lambda_7, \lambda_8) \in \{0,1\}^8 \end{array}$$

In general, let column q be represented by an index vector $a_q \in \{0,1\}^m$ corresponding to a possible assignment of items to a bin (i.e. $\sum_i a_{i\,q} \leq C$). Let Q be the set of all such feasible assignments. The reformulation is of the form

$$\min \quad \sum_{q \in Q} \lambda_q$$

$$
\begin{array}{rcll}
\text{s.t.} & & & \\
\sum_{q \in Q} a_{i\,q} \lambda_q & = & 1 & \forall\, i \\
\lambda_q & \in & \{0,1\} & \forall\, q \in Q.
\end{array}
$$

The cardinality of the set Q (and thus the number of variables in the formulation) depends on the data, but it can be very large. For instance, if $s_i \leq \frac{C}{n}\, \forall i$, then $|Q|$ is greater or equal to $\begin{pmatrix} \text{m} \\ \text{n} \end{pmatrix}$ (i.e. the number of different subsets of n elements from among a set of m elements).

How integer programming formulations with many variables arise

In many real-life situations, one can view the optimization problem as selecting a few subsets from a very wide choice of subsets (representing possible scenarios, proposals, or patterns), that together satisfy some requirements at minimal cost (maximal profit). In such cases, one can formulate the optimization problem directly as an integer program with a large number of variables.

Alternatively, extensive formulations are often obtained as the result of the decomposition of the problem constraint set. Given an integer problem formulation of the form

$$
\begin{array}{rcl}
\min & & cx \\
\text{s.t.} & & \\
Ax & = & b \\
Dx & \leq & d \\
x & & \text{integer}
\end{array}
$$

whose constraints are partitioned into a class of global constraints ($Ax = b$) and a class of specific constraints referred to as a subsystem ($Dx \leq d$ and x integer), one can potentially enumerate all the solutions of the subsystem of constraints and operate a variable transformation that consists of reformulating the problem in terms of the subsystem solutions. For instance, one can partition the constraints of the bin packing problem by considering constraints (1) as the global constraints ($Ax = b$), and constraints (2) as a subsystem of constraints ($Dx \leq d$).

There are two kinds of situation where the decomposition approach is naturally used. One is when the optimization problem is easy to solve over the subsystem of constraints (i.e. $Dx \leq d$ are nice constraints because they model a well-known

polyhedron, for instance, a network flow problem), but constraints $Ax = b$ complicate the problem. The second and most important case for us is that in which the problem is decomposable, i.e. constraints $Dx \leq d$ have a block diagonal structure (see Figure 1), so that the subsystem decomposes into K subsystems. The global constraints are then called linking constraints, since the K blocks associated with the K subsystems would be independent without the presence of the constraints ($Ax = b$). In the bin packing example, $Dx \leq d$ decomposes into one subsystem of constraints for each bin.

Figure 1: Structure of the constraint matrix.

In some applications, the only way to formulate the problem is to use a large number of variables (see Barnhart (1994) [4]). Whether the integer program with a large number of variables results from decomposition or not, we call it the *master* by reference to the Danzig-Wolfe decomposition principle in linear programming.

Dealing with a formulation with a large number of columns

The master integer program is solved to optimality by branch-and-bound. To carry out the implicit enumeration scheme, we use approximations provided by the linear programming relaxations of the master (master LP). Typically the columns of the master are not known explicitly or/and are too many to be written out explicitly. In practice, one works with a linear programming relaxation involving a small subset of columns and generate new ones if and when needed. Since an optimum solution typically involves only a few columns, the idea is to generate the columns that are potentially part of the optimum solution and to avoid considering those that are not attractive.

In practice, during the LP optimization procedure, one solves an optimization sub-

problem called the column generation subproblem to evaluate the attractiveness of columns that have not yet been included in the formulation. If one or more columns can potentially improve the current solution, they are added to the formulation and the procedure reiterates. This technique is known as *column generation.* The master formulation with many columns is also called the *column generation formulation.* The combination of column generation with branch-and-bound is referred to as *IP column generation.*

Interest of a formulation with large number of variables

Our motivation for considering integer programs with a large number of variables is twofold. First, we want to solve a telecommunication problem consisting of the allocation of the telecommunication traffic to an existing network. This problem can be formulated as a compact integer program. However, the standard LP based branch-and-bound algorithm cannot solve realistic sized instances of this problem. Indeed, the linear programming relaxation bound is a poor approximation of the integer optimal value. Moreover, standard branch-and-bound fails due to the inherent symmetry of the problem. Exploiting the constraint matrix structure, we decompose the problem. The master reformulation provides a tighter bound and eliminates the difficulties associated with the problem symmetry. Thus, this problem is a good candidate for the IP column generation solution method.

In many other applications, the master reformulation is advantageous because its LP relaxation provides a tight bound on the integer solution value. Moreover, it eliminates difficulties inherent in the symmetric structure of the problem and it offers a natural 2-level decomposition with an aggregated level: the master, and a disaggregated level: the subproblem defined by the subsystem of constraints.

The second motivation for studying the column generation approach for integer programming is the recent success of this technique. In particular, Aghezzaf et. al. (1992) [1] have shown that, for the problem of packing subtrees of a tree, the master LP formulation has an integer optimal solution. This raises an interesting question: When does the decomposition approach yields good approximations? In the next Section, we recall the theoretical conditions under which the master LP solves the integer problem. But, when these conditions do not hold, does the master LP provide good bounds? In particular, how do these bounds compare with the ones obtained with the polyhedral approach? More generally, is the column generation approach robust across applications? These questions go beyond the scope of our study which

is mainly concerned with the development of an IP column generation algorithm. However, in the course of our computational experiments, we have tested the column generation approach on two other types of problems for which the polyhedral approach performs well: graph partitioning problems and single-machine multi-item lot-sizing problems. These tests partially address the question of robustness and they may be compared with the existing results obtained using the polyhedral approach (see [7], [17]).

1.2 Dantzig-Wolfe Decomposition in Linear Programming

The decomposition principle was introduced by Dantzig and Wolfe in 1960 [11]. It leads to the use of a column generation algorithm to solve large scale linear programs. This decomposition approach has become standard linear programming methodology and is described in numerous textbook and publications (see for instance , [21], [32], [40], and [43], on which we have based the presentation below).

Dantzig-Wolfe decomposition was originally introduced in order to solve large scale linear programming problems on computers with limited core storage capacity. It was not meant to compete with the simplex method but rather to complement it by extending the range of applicability of linear programming. This probably explains why it is sometimes referred to as generalized linear programming. When applied to linear programs whose coefficients matrices have the structure presented in Figure 1, i.e., one or more independent blocks (subsystems) linked by coupling equations, it operates by forming an equivalent *master* linear program having only a few more rows than there are linking constraints in the original formulation but having many more columns.

Consider a general linear programming problem of the form

$$\begin{array}{rcl} \min & & cx \\ \text{s.t.} & & \\ Ax & = & b \\ Dx & \leq & d \\ x & \geq & 0 \end{array}$$

where $Ax = b$ represent the linking constraints. Let $\{p_q\}_{q \in Q}$ be the set of extreme points of the polyhedron $\{x \in I\!R^n_+ : Dx \leq d\}$, assuming that it is bounded and

non-empty. Then the equivalent master linear program is of the form

$$\begin{aligned} \min \quad & \sum_{q \in Q} c_q \lambda_q \\ \text{s.t.} & \\ \sum_{q \in Q} a_q \lambda_q &= b \\ \sum_{q \in Q} \lambda_q &= 1 \\ \lambda_q &\geq 0 \qquad \forall q \in Q. \end{aligned}$$

where $c_q = c\, p_q$ and $a_q = A\, p_q$.

In most applications, solving the master linear program directly is totally impractical because of the enormous number of variables in the formulation. To overcome this difficulty the master linear program is solved using a column generation procedure, which we describe in detail in Chapter 3. In short, the idea of a column generation algorithm is to work with just a subset of columns (variables) and generate missing ones if and when needed. At a given iteration of the column generation algorithm, one solves a restricted master linear program over a subset $\tilde{Q} \subseteq Q$ of columns. Letting $(\pi, \mu) \in I\!R^m \times I\!R$ be an optimal dual solution, one checks if the current master LP solution is optimal by solving the *pricing subproblem*:

$$v(\pi, \mu) = \min\{(c - \pi A)x : \; Dx \leq d, \; x \geq 0\} \; + \; \mu.$$

If $v(\pi, \mu) \geq 0$, linear programming theory tells us that the master LP is solved. Otherwise, the solution of the subproblem x^* defines a new column, $a_q = A\, x^*$ and $c_q = c\, x^*$, that is added to the restricted master formulation.

A well-known difficulty with the column generation procedure is the typically large number of iterations required to prove LP optimality. This drawback is referred to as the *tailing-off effect*. However, recent studies (see for instance Goffin et. al. (1992) [24]) show that tailing-off does not systematically happen in linear programming.

The decomposition methodology is also called the price directive decomposition method for its important economic interpretation as a coordination mechanism for decentralization of large organizations. If each subsystem (division) works independently of the others, then we might assume that it adjusts the levels of its operations so as to minimize its own cost. However, the subsystems are not independent, but linked by the constraints on the use of resources shared on a global level. The goal

of a coordinator is to set prices π on the shared resources (the right hand side of the linking constraints) so that each of the K divisions can independently optimize their own operations. The rationale for selecting prices is to assure that the resource constraints are satisfied on the global level. The interpretation of the master is that the coordinator makes the best use he can of the information he has about the operating possibilities of each division in minimizing total cost.

1.3 Decomposition and Reformulation of an Integer Program

Although the decomposition principle was developed for linear programming, it is commonly applied in integer programming to derive good approximations. In this context, a linear programming reformulation (named the master problem) was proposed as an alternative to the standard linear relaxation because it leads to a generally tighter relaxation. Thus, the decomposition approach remained essentially a linear programming concept.

Here, we present the decomposition in integer programming terms, showing that the reformulation of the initial compact integer program is an integer program involving many *integer* variables. In other words, when we apply decomposition to an integer program, we obtain a *master integer program* with a large number of columns. The linear relaxation of the master is then solved using a column generation algorithm. The column generation subproblem is also an integer program. We emphazise the equivalence between the master integer programming reformulation and the initial compact integer programming formulation. Solving the master IP resulting from the decomposition is equivalent to solving the original integer programming formulation in the same way that, in linear programming, a solution of the master leads to an optimal solution to the original formulation.

A General Model $[P]$

The general model we consider can be formulated as a nonlinear integer program $[P]$:

$$
\begin{array}{lll}
 & \min & c(x) \\
[\mathrm{P}] & \text{s.t.} & \\
 & Ax = b & \\
 & x \in X &
\end{array}
$$

where (A, b) is an integer $m \times (n+1)$ matrix, $X = \{x \in I\!N^n : Dx \leq d\}$, and (D, d) is an integer $l \times (n+1)$ matrix. In practice, we will work with applications in which matrix D has the block diagonal structure presented in Figure 1.

We make the restrictive assumption that the cost function can be linearized by introducing auxiliary integer and continuous variables $(y, z) \in I\!N^p \times I\!R^r$. The resulting augmented problem $[P']$ is of the form:

$$[P'] \qquad \begin{array}{lll} \min & fx + gy + hz & \\ \text{s.t.} & & \\ Ax & = & b \\ (x, y, z) & \in & S \end{array}$$

where f, g and h represent the cost vector, $S \subseteq I\!N^n \times I\!N^p \times I\!R^r$ describes the combinatorial restrictions on x and the link between the x variables and the auxiliary variables (y, z). We assume that S can be represented as a mixed integer program. Moreover, $c(x) = \min_{(y,z)}\{fx + gy + hz : \ (x, y, z) \in S\}$ and $X = proj_x S$.

The Reformulation of an IP by Decomposition

The fundamental theoretical principle, on which the integer programming decomposition is based, is that the set of integer points in a polyhedron can be finitely generated. To derive an IP master formulation, we use the following result concerning the set of integer points (see Section I.4.6 of Nemhauser and Wolsey (1988) [40]).

Proposition 1 *If $F = \{x \in I\!R^n_+ : \ Dx \leq d\} \neq \emptyset$ and $X = F \cap I\!N^n$ where (D, d) is an integer $l \times (n+1)$ matrix, then the following statements are true:*
(i) There exist a finite *set of points $\{p_q\}_{q \in Q}$ of X and a* finite *set of rays $\{r_j\}_{j \in J}$ of F such that*

$$X = \{x \in I\!R^n_+ : \ x = \sum_{q \in Q} \lambda_q p_q \ + \ \sum_{j \in J} \beta_j r_j, \ \sum_{q \in Q} \lambda_q = 1, \ \lambda \in I\!N^{|Q|}, \ \beta \in I\!N^{|J|}\}. \qquad (3)$$

(ii) If F is a cone ($d = 0$), there exists a finite *set of of rays $\{r_h\}_{h \in H}$ of F such that*

$$X = \{x \in I\!R^n_+ : \ x = \sum_{h \in H} \gamma_h r_h, \ \gamma \in I\!N^{|H|}\}. \qquad (4)$$

Thus X can be generated by taking a point p_q for some q in the finite set Q plus a non-negative integer combination of extreme rays of F. When X is bounded, as we

shall assume from now on for simplicity, the finite set of points $\{p_q\}_{q\in Q}$ is the set X itself.

Using this characterization of integer polyhedra, in place of the Minkowski's theorem that is the basis for Dantzig-Wolfe decomposition in linear programming, we obtain an integer programming decomposition. To parallel the developments leading to the master formulation in linear programming, we state that any point of X is of the form:

$$\begin{aligned} x &= \sum_{q\in Q} \lambda_q p_q \\ \text{s.t.} & \\ \sum_{q\in Q} \lambda_q &= 1 \\ \lambda_q &\in I\!N \qquad\qquad \forall q \in Q \end{aligned}$$

Note that this is a trivial statement because we have assumed that X is bounded, but this statement indicates how to obtain a reformulation when X is unbounded.

Replacing x by its equivalent expression in problem $[P]$ leads to the formulation

$$\begin{aligned} \min \quad & c(\sum_{q\in Q} \lambda_q p_q) \\ \text{s.t.} & \\ A\ (\sum_{q\in Q} \lambda_q p_q) &= b \\ \sum_{q\in Q} \lambda_q &= 1 \\ \lambda_q &\in I\!N \qquad\qquad \forall q \in Q. \end{aligned}$$

Let us define $c_q = c(p_q)$ and $a_q = Ap_q$. Note that the convexity constraint together with the integrality constraints amounts to imposing $\lambda_q \in \{0, 1\}$. So, in any feasible solution, only one λ_q will take value 1, say it is λ_l, while all the others take value zero. Thus, for any feasible solution of $[P]$, $c(\sum_{q\in Q} \lambda_q p_q) = c(p_l) = c_l = \sum_{q\in Q} \lambda_q c_q$, and the above formulation is equivalent to

$$\begin{aligned} \min \quad & \sum_{q\in Q} c_q \lambda_q \\ \text{s.t.} & \\ \sum_{q\in Q} a_q \lambda_q &= b \end{aligned}$$

$$\begin{aligned} \sum_{q \in Q} \lambda_q &= 1 \\ \lambda_q &\in \{0,1\} \qquad \forall q \in Q, \end{aligned}$$

which we refer to as the *master* problem.

Solving the master binary integer program is equivalent to solving the original integer formulation $[P]$. However, the linear programming relaxations of these two formulations are different. As we shall show in the Section 1.4, the master linear programming relaxation generally leads to a tighter lower bound than the linear relaxation of the original formulation.

We point out that the decomposition of an integer program is sometimes introduced differently in the literature (see [4], [16]). The alternate presentation uses the fact that the integer programming optimization of the subproblem is equivalent to a linear programming optimization over the convex hull of the subproblem extreme points (see Nemhauser and Wolsey (1988) [40]). That is, any point of $conv(X)$ can be expressed as $\sum_{q \in \hat{Q}} \alpha_q \hat{p}_q$ with $\sum_{q \in \hat{Q}} \alpha_q = 1$ and $\alpha_q \geq 0, \ \forall\, q \in \hat{Q}$, where $\{\hat{p}_q\}_{q \in \hat{Q}}$ is the set of extreme points of $conv(X)$.

This approach yields a slightly different master IP formulation:

$$\begin{aligned} \min \quad & \sum_{q \in \hat{Q}} c_q \alpha_q \\ \text{s.t.} \quad & \\ \sum_{q \in \hat{Q}} a_q \alpha_q &= b \\ \sum_{q \in \hat{Q}} \alpha_q &= 1 \\ \sum_{q \in \hat{Q}} \alpha_q \hat{p}_q &\in I\!N^n \\ \alpha_q &\geq 0 \qquad \forall\, q \in \hat{Q}. \end{aligned}$$

Here the integrality constraints are expressed in terms of the original variables $x = \sum_{q \in \hat{Q}} \alpha_q \hat{p}_q$. Note that imposing integrality of the α's would not lead to an equivalent integer programming reformulation as the optimal integer solution of $[P]$ may be a interior point of X.

When the x variables are binary, $\hat{Q} = Q$ and the two master reformulations are equivalent. However, in the case of general integer variables, they are different. In

this thesis, we shall present an integer programming column generation algorithm that encompasses the case of a set X containing general integer points. Through the branching scheme and the associated subproblem modification scheme, one can generate interior points of X as solutions of the pricing subproblem.

The Case of a Decomposable Problem

When problem $[P]$ has a structured constraint matrix like the one presented in Figure 1, we say that it is decomposable. Then, the pricing subproblem decomposes into K independent subproblems: i.e. $X = X^1 \times \ldots \times X^K$.

If we let $\{p_q^k\}_{q \in Q^k}$ represent the set of feasible solutions in X^k, then for all $x^k \in X^k$, there exists $\lambda^k \in \{0,1\}^{|Q^k|}$ such that $x^k = \sum_{q \in Q^k} \lambda_q^k p_q^k$ and $\sum_{q \in Q^k} \lambda_q^k = 1$. The resulting master is:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{K} \sum_{q \in Q^k} c_q^k \lambda_q^k & & \\
\text{s.t.} & & & \\
\sum_{k=1}^{K} \sum_{q \in Q^k} a_q^k \lambda_q^k &= b & & \\
\sum_{q \in Q^k} \lambda_q^k &= 1 & & \forall k \\
\lambda_q^k &\in \{0,1\} & & \forall k,\ q \in Q^k.
\end{aligned}$$

where $c_q^k = c(p_q^k)$ and $a_q^k = A^k p_q^k$. In addition, there is a pricing subproblem associated with each subsystem k.

When the K subsystems are identical, as often happens in practice, the master formulation becomes a general integer program. If $A^1 = \ldots = A^K$ and $X^1 = \ldots = X^K = X = \{p_q\}_{q \in Q}$, let $\lambda_q = \sum_{k=1}^{K} \lambda_q^k$. Then the master take the form

$$\begin{aligned}
\min \quad & \sum_{q \in Q} c_q \lambda_q & & \\
\text{s.t.} & & & \\
\sum_{q \in Q} a_q \lambda_q &= b & & \\
\sum_{q \in Q} \lambda_q &= K & & \\
\lambda_q &\in \{0, \ldots, K\} & & \forall\, q \in Q.
\end{aligned}$$

If the null vector is a solution of X, it may be omitted from the formulation and the aggregated convexity constraint may be replaced by $\sum_{q \in Q} \lambda_q \leq K$.

The Example of the Cutting Stock Problem

Gilmore and Gomory (1961-1963) [22] [23] present a first application of Dantzig-Wolfe LP decomposition to an integer program for the cutting stock problem. They used it to obtain a very tight lower bound and very good heuristic solutions. The bin packing problem we introduced in Section 1.1 is a special case of the cutting stock problem that we present now.

Suppose that a paper (textile) company has a supply of large rolls of paper of width R, but that customer demand is for paper of smaller width. Suppose b_i rolls of strip of width $l_i \leq R$, $i = 1, ..., m$ need to be produced. We obtain smaller rolls by slicing a large roll using a particular pattern. Invariably, the cutting process involves some waste. The company would like to minimize waste, or equivalently, to meet demand using the fewest number of rolls.

One way to formulate this problem (for the purpose of exposition) is to use the following variables:
$y^k = 1$ if roll k is used, 0 otherwise and
$x_i^k =$ the number of pieces of width l_i cut in roll k.
Let K be an upper bound on the number of rolls used in any optimal solution. Then, the problem of minimizing the number of rolls, while satisfying demand, can be formulated as:

$$\begin{aligned}
\min \quad & \sum_{k=1}^{K} y^k & & \\
\text{s.t.} & & & \\
\sum_{k=1}^{K} x_i^k &= b_i & & \forall\, i \\
\sum_i l_i x_i^k &\leq R\, y^k & & \forall\, k \\
x_i^k &\leq b_i\, y^k & & \forall\, i,\ k \\
x_i^k &\in I\!N & & \forall\, i,\ k \\
y^k &\in \{0,1\} & & \forall\, k
\end{aligned}$$

Let $S = \{(x, y) \in I\!N^m \times \{0,1\} : \sum_i l_i x_i \leq Ry$, and $x_i \leq b_i y\ \forall i\}$. Then, $c(x) = \min_y\{y : (x, y) \in S\}$, that is $c(x) = 0$ if $(x, y) = (0, 0)$ and $c(x) = 1$ otherwise, and $X = proj_x\, S = \{x \in I\!N^m : \sum_i l_i x_i \leq R\}$. The set of feasible solutions of X is just the set of cutting patterns that satisfy the knapsack constraint. Since the matrix A^k associated with subsystem k is a unit matrix, there is no distinction between the points of X, $\{p_q\}_{q \in Q}$, and the columns of the master, a_q (a_q represents a cutting

pattern in which $a_{i\,q}$ pieces of width l_i are cut from a paper roll of width R). The cutting pattern a_q is feasible if $\sum_i l_i a_{i\,q} \leq R$. The associated cost $c(a_q)$ is 1 if a_q is non null. If one does not include the null pattern in Q, the master integer program is

$$\begin{array}{rcll} \min & & \displaystyle\sum_{q\in Q} \lambda_q & \\ \text{s.t.} & & & \\ \displaystyle\sum_{q\in Q} a_{i\,q}\lambda_q & = & b_i & \forall\, i \\ \displaystyle\sum_{q\in Q} \lambda_q & \leq & K & \\ \lambda_q & \in & I\!N & \forall\, q \in Q. \end{array}$$

Given the cost structure, the aggregated convexity constraint may be dropped.

The latter formulation (without the convexity constraint) is frequently used in the literature to present the cutting stock problem. The master IP formulation with a large number of columns is equivalent to the more compact formulation we started with. We usually refer to the compact formulation as the original or the initial formulation. However, one can also view the master formulation with a large number of variables as the natural starting formulation instead of considering it as the result of a variable transformation. This is generally the case for the cutting stock problem. In Chapter 3, where we present a solution method for the IP master, we shall adopt the latter point of view, starting immediately from a master IP formulation which involves a large number of columns.

1.4 Alternative Relaxations of an Integer Program

In this Section, we present basic theoretical results. We show that typically the master reformulation of $[P]$ has a stronger LP relaxation than the initial more compact formulation. In the process, we discuss several solution methods for problem $[P]$ that lead to equivalent relaxations, i.e., relaxations that produce the same bound as the master LP. These results are well-known but important to remember.

The master Linear Relaxation

The linear programming relaxation of the master is of the form

$$Z_{MLP} = \min \quad \sum_{q\in Q} c_q \lambda_q$$

$$
\begin{aligned}
& \text{s.t.} \\
\sum_{q \in Q} a_q \lambda_q &= b \\
\sum_{q \in Q} \lambda_q &= 1 \\
\lambda_q &\geq 0 \qquad\qquad \forall q \in Q.
\end{aligned}
$$

where $c_q = \min_{(y,z)}\{fp_q + gy + hz : (p_q, y, z) \in S\}$, and $a_q = Ap_q$ for some point $p_q \in X = proj_x\, S = \{p_q\}_{p \in Q}$. Let $S_{\min} = \{(x, y^*, z^*) \in S : (y^*, z^*) = argmin_{(y,z)}(fx + gy + hz)\}$. Then each point $p_q \in X$ is the projection on the x-space of a point $(p_q, y_q, z_q) \in S_{\min}$ such that $c_q = fp_q + gy_q + hz_q$. Moreover,

$$
\begin{aligned}
Z_{MLP} = \min \quad & fx + gy + hz \\
\text{s.t.} & \\
Ax &= b \\
(x, y, z) &= \textstyle\sum_{q \in Q} \lambda_q \, (p_q, y_q, z_q) \\
\sum_{q \in Q} \lambda_q &= 1 \\
\lambda_q &\geq 0 \qquad\qquad \forall q \in Q.
\end{aligned}
$$

That is

$$
Z_{MLP} = \min\{fx + gy + hz : Ax = b,\ (x, y, z) \in conv(S_{\min})\}
$$

Note that replacing $conv(S_{\min})$ by $conv(S)$ lead to the same value of Z_{MLP}.

In comparison, the formulation of the original integer program and its linear relaxation are:

$$
\begin{aligned}
Z_{IP} &= \min\{fx + gy + hz : Ax = b,\ (x, y, z) \in S\}, \\
Z_{LP} &= \min\{fx + gy + hz : Ax = b,\ (x, y, z) \in S_{LP}\},
\end{aligned}
$$

where S_{LP} is the linear relaxation of S, i.e. the feasible set obtained after removing all the integrality constraints.

Because $S_{LP} \supseteq conv(S) \supseteq S$,

$$
Z_{LP} \leq Z_{MLP} \leq Z_{IP}
$$

Moreover, if $S_{LP} = conv(S)$, i.e. if the polyhedron S_{LP} has integer extreme points (which is known as the integrality property), then $Z_{LP} = Z_{MLP}$. On the other

hand, if $conv(\{(x,y,z) \in S : Ax = b\}) = \{(x,y,z) \in conv(S) : Ax = b\}$, then $Z_{MLP} = Z_{IP}$.

Lagrangean Relaxation

Dualizing the linking constraints ($Ax = b$) leads to the so-called Lagrangean subproblem:

$$L(\pi) \quad = \quad \min\{(f - \pi A)x + gy + hz + b\pi : (x,y,z) \in S\},$$

where π represent the Lagrange multipliers associated with the dualized constraints. $L(\pi)$ defines a lower bound on Z_{IP}. The best possible such lower bound is obtained by solving the Lagrangean dual problem

$$Z_{LD} \quad = \quad \max_{\pi} L(\pi).$$

One common approach to solve the Lagrangean dual is to use a subgradient algorithm.

It is well-known that the Lagrangean dual resulting from the relaxation of the linking constraints ($Ax = b$) provides a relaxation equivalent to (giving the same bound as) the master LP. Geoffrion (1974) [21] proved that the Lagrangean dual linear program and the master linear program are dual problems (see also Nemhauser and Wolsey (1988) [40]).

In Section 2.1, we illustrate the equivalence between the Lagrangean dual to the master LP formulation for a specific application. Barnhart et.al.(1994) [4] summarize the pros and cons of the Lagrangean duality approach versus the column generation approach.

A Cutting Plane Approach

A third approach, known as partial convexification, leads to another equivalent relaxation. It consists of approximating $conv(S)$ by adding cuts to S_{LP}. Suppose that we have an implicit representation of $conv(S)$ in terms of linear inequalities, then, we can solve the following separation problem: given a point $(x,y,z) \notin conv(S)$, produce one or more violated valid inequalities. A cutting plane algorithm consists of iteratively adding violated inequalities to the formulation of S_{LP}. Let S^t_{LP} be the strengthened formulation of S_{LP} at iteration t of the cutting plane algorithm, then

$$Z^t_{CP} \quad = \quad \min\{fx + gy + hz : Ax = b, (x,y,z) \in S^t_{LP}\},$$

with $S_{LP} \supseteq S_{LP}^t \supseteq S_{LP}^{t+1} \supseteq conv(S)$. If the separation algorithm used is exact, i.e. a violated inequality is found whenever the current solution $(x, y, z)^t \notin conv(S)$, the algorithm terminates with a solution to

$$Z_{CP} = \min\{fx + gy + hz : Ax = b, (x, y, z) \in conv(S)\}.$$

Comparison of the 3 approaches

We have presented three different relaxations and algorithms to solve these relaxations: the master LP relaxation solved by the column generation algorithm, the Lagrangean relaxation solved using a subgradient algorithm, and the partial convexification relaxation (the polyhedral approach) solved using a cutting plane algorithm. They all lead to the same bound

$$Z_{MLP} = Z_{LD} = Z_{CP}$$

However, they differ in terms of assumptions, approximation strategies, and convergence.

The decomposition approach and the Lagrangean dual approach both lead to a subproblem that is to optimize over the set S. In contrast, the cutting plane approach leads to a subproblem that is to solve the separation problem over S.

During the column generation algorithm, primal feasibility is maintained. So the intermediate feasible primal solutions define upper bounds on Z_{MLP}. In comparison, the polyhedral approach is a dual approximation approach in which the intermediate solutions are not primal feasible. The intermediate bounds are valid lower bounds increasing toward Z_{CP}.

During the subgradient algorithm used to solve the Lagrangean dual, the intermediate solutions may not be primal feasible. The intermediate bounds are valid lower bounds converging towards Z_{LD}. Note that, since the Lagrangean subproblem and the column generation subproblem are identical, Lagrangean bounds can typically be used as intermediate lower bounds during the column generation process.

The column generation algorithm converges monotonically and finitely, while the subgradient algorithm is not monotonic and might require an infinite number of iterations to terminate. With the cutting plane algorithm, finite convergence is not

guaranteed unless one uses facet defining inequalities of S, but the convergence is monotonic.

1.5 Literature Overview

In the last 30 years, the development of general integer programming (IP) techniques has lead to an extension of the range and size of integer programs that can be solved to optimality. A particular effort has been made to obtain tight relaxations of integer programs. Techniques such as Lagrangean relaxation, problem convexification (polyhedral approach), and problem decomposition and reformulation have been developed intensively. The next step has been to embed these bounding techniques in an implicit enumeration approach to solve integer programs to optimality more efficiently than with standard branch-and-bound.

The first experiment along these lines has concerned the Lagrangean approach which is probably the easiest to implement. Held and Karp (1970 -1971) [26]-[27] have successfully applied Lagrangean bounding techniques to the traveling salesman problem. In the last decade, the polyhedral approach has been used extensively (see for instance Balas et. al. (1993) [2]). The integration of the cutting plane algorithm within a branch-and-bound procedure has been named branch-and-cut (for general exposition see Hoffman and Padberg (1985) [28] and Nemhauser and Wolsey (1988) [40]). It is only recently that difficult mixed integer programs have been solved to optimality using the decomposition approach to produce good bounds.

Although column generation formulations of integer programs have been proposed and discussed for several decades, for many years this technique was used to produce solutions that were good but not always optimal. The pioneering work of Gilmore and Gomory (1961-1963) [22] [23] on the cutting stock problem demonstrates the strength of the linear programming relaxation of the master formulation involving a large number of columns. Marcotte (1985) [34] shows that the cutting stock optimum LP solution, once rounded so as to satisfy the integrality constraints, provides a very satisfactory solution of the integer problem. Minoux (1987) [36] shows how several important combinatorial optimization problems can be reformulated and tackled by column generation. Johnson et. al. (1993) [29] use a column generation approach to solve a graph partitioning problem (clustering problem). When they obtain a fractional master LP solution, they proceed to solve the restricted master integer program (i.e. the formulation containing only the columns that have been

generated) by branch-and-bound without generating any new columns. With this heuristic procedure, they obtain a provably optimal integer solution in 9 out of 12 instances.

The use of column generation to solve integer programs to optimality is recent. Large airline staffing problems known as crew scheduling problems as well as multi-commodity network flow problems have been tackled by column generation, for instance see Barnhart et. al. (1991) [3] and [5]. Degraeve (1992) [12] considers some production scheduling problems. In [44], Vance et. al. (1992) propose an exact optimization procedure for the binary cutting stock problem. Savelsbergh (1993) [41] treats the generalized assignment problem. Numerous applications in routing and scheduling are covered in a recent survey of Desrosiers et.al. (1994) [16]: the aircraft fleet assignment problem, the pickup and delivery problem, the urban transportation problem, the vehicle routing problem with time windows, etc. In a recent survey, Barnhart et. al. (1994) [4] present an unified review of general IP column generation ideas that have appeared in different contexts.

1.6 Content of the thesis

The main aspect of this work is the study of column generation as a tool to solve large scale integer programs to optimality. We implement an algorithm combining column generation and branch-and-bound. We generalize previous results to extend the class of problems that can be dealt with using this approach. We test the algorithm for three types of problems: a telecommunication network design problem plus a variant of the same problem, a graph partitioning problem and a single-machine multi-item lot-sizing problem. We share the insights of our computational experiments.

Both column generation and branch-and-bound are common techniques that have been applied in integer programming for many years. Embedding column generation into branch-and-bound sounds like a simple combination of well-known procedures. In fact, it is not straightforward. The branch-and-bound implicit enumeration mechanism interferes with the column generation mechanism. Moreover, as the column generation procedure is repeated at each node of the branch-and-bound tree, it becomes very important to perform it efficiently. Below we mention how these issues have been dealt with previously and point out our contributions.

The first concern in an IP column generation algorithm is how to eliminate fractional solutions. Conventional integer programming branching based on variable fixing is not straightforward to implement in a column generation context. Indeed, fixing a master variable to zero corresponds to forbidding the use of a particular column. However, this column will probably reappear as the solution of the column generation subproblem. Then, one could generate the 2^{nd} best solution of the column generation subproblem (deeper in the branch-and-bound tree, one might end up searching for the n^{th} best solution). This technique has been used in [25] and [37]. A second approach consists of adding constraints to the column generation subproblem in order to make the targeted column infeasible. However, it may destroy the structure of the column generation subproblem. For instance, in the context of the cutting stock problem, it eliminates the possibility of solving the subproblem exactly by dynamic programming.

Alternate branching schemes have been explored. In their research on routing and scheduling problems, Desrosiers et.al. (1994) [16] propose to take the branching decision in the initial compact formulation as opposed to the master reformulation and to reapply the Dantzig-Wolfe decomposition scheme to the augmented initial formulation to define a new master LP formulation at each node of the branch-and-bound tree.

In a recent survey on solving integer programs by column generation, Barnhart et. al. (1994) [4] recall an earlier result of Ryan and Foster (1981) [38] leading to a branching scheme in terms of the master variables. This method for branching is valid for a 0-1 set partitioning formulation with right-hand-sides equal to 1. It consists of selecting two rows of the master formulation. Then, all columns that have an entry one in both of the chosen rows are eliminated on one branch, whereas, on the other branch, all columns that have an entry one in only one of the rows are eliminated. This branching scheme has a natural interpretation in the initial compact formulation.

The branching scheme we propose in this thesis includes the Ryan and Foster branching scheme and more. It allows us to present an unified view of various simple branching rules. Moreover, it provides a way of dealing with an extension of the set partitioning problem in which the variables as well as the right-hand-sides are general integers. We even extend the scheme, in theory, to a model with general integer matrix entries, which includes the general integer cutting stock problem.

The column generation algorithm which is applied at each node of the branch-and-bound tree requires one to solve the master linear program and the subproblem integer program at each iteration. Due to the typically large number of column generation iterations often needed to prove LP optimality, it is important to perform each iteration as efficiently as possible, or/and to limit the number of iterations (i.e. the tailing-off effect) as much as possible.

Barnhart et.al. (1994) [4] propose two alternative ways to improve the efficiency in solving the LP's: employ specialized simplex procedures that exploit the problem structure, and alter the master problem formulation to reduce the number of columns. We have not yet tried to incorporate these enhancements in our algorithm.

In some applications, the subproblem solution is not too computationally intensive and most of the algorithm effort is spent on solving the master LP. With other applications, such as those we treat in this thesis, the computational bottleneck is the solution of the subproblem. The column generation subproblem is a difficult integer program. In the first iterations of the column generation algorithm, we use a heuristic procedure to generate columns. But, as column generation approaches to LP optimality, it becomes harder to find negative reduced cost columns heuristically and an exact optimization procedure is required to generate new columns. Thus, it is crucial to control tailing-off. To this purpose, we propose a bounding scheme based on Lagrangean duality and we derive conditions for early termination of the column generation procedure. Moreover, we show that these termination conditions can be expressed in terms of a priori bounds on the column generation subproblem value.

The performance of the IP column generation algorithm may be improved by judicious implementation choices. In this thesis, we describe some enhancements of the procedure. We discuss the formulation we use in practice, the way we branch, the column generation subproblem solution, the column selection criteria, and more. In presenting our computational tests, we try to bring some understanding on the impact of different implementation strategies.

Our computational results highlight the attractiveness of the column generation approach. The bounds provided by the master LP solution at the root node are shown to be tight. The embedding of this bounding technique in branch-and-bound allows

one to solve realistic size problems to optimality within reasonable time. Moreover, experiments with different applications partially address the question of robustness of the method and the comparison with other integer programming techniques.

1.7 Chapter Outline

The rest of this thesis is organized in four Chapters in which we discuss respectively the applications we consider, the IP column generation algorithm, its implementation, and computational results.

In Chapter 2, we describe the problems that we will try to solve. We apply the decomposition principle to reformulate these problems as integer programs with a large number of variables.

In Chapter 3, we describe the theoretical aspects of the exact optimization procedure that solves integer programs with a large number of variables. We present the algorithm for a general set partitioning model that includes all the applications we consider. We describe the column generation procedure used to solve the master linear relaxation at a node of the enumeration tree and we present the branch-and-bound procedure with emphasis on the branching scheme and the efficient use of bounds.

Chapter 4 contains the details concerning a practical implementation of the IP column generation algorithm. We discuss the choice of a master formulation (partitioning vs covering). We say how we start the column generation procedure, how we solve the subproblem, and how we select a branching rule. We discuss some column selection strategies. We then give a global picture of the algorithm.

Finally, in Chapter 5, we present the results of our computations for the different applications. We also provide some computational comparative test results that show the efficiency of our column generation early termination technique and justify our implementation choices. In the conclusions, we summarizes the pros and cons of IP column generation and we make suggestions for further research.

2 Decomposition of Integer Programs: 4 applications

The motivation for this thesis has been to solve real-life problems which happen to be well suited for decomposition. In this Chapter, we shall describe the four applications we have been working on.

2.1 A Telecommunication Network Design Application

Given a graph $G(V, E)$ that represents a city, in which a vertex (node) i represents a telecommunications center and an edge e represents a pair of points (centers) in the city between which there is a positive demand d_e for telecommunication traffic. The problem (P_{ND}) is to assign traffic on an existing network which is made of fiber optic cycles, called rings. If the traffic on a given edge e is assigned to ring k, some costly piece of equipment (called a multiplexer) has to be installed at the end points of e to make the connection between the center and the ring network. Once a multiplexer is installed at a center (node), it can be shared with other edges incident to this node that are assigned to the same ring. However, the limited capacity of a multiplexer implies an upper bound C on the total traffic that can be assigned to a ring.

The objective is to minimize the number of multiplexers needed to realize the connections. Thus one tends to assign edges sharing a node to the same ring. If feasible, the optimal solution would be to assign all edges on the same ring (needing a total of $|V|$ multiplexers). But the limited ring capacity restricts the assignment possibilities.

A small instance of this problem is given in Figure 2, in which $V = \{A, B, C, D\}$, $d_{AB} = 10$, $d_{AC} = 12$, $d_{AD} = 14$, $d_{BC} = 16$, $d_{BD} = 18$, and $d_{CD} = 20$. Figure 3 represents a feasible solution when the capacity is 60. Next, we present the model and we discuss its difficulty. Then, we derive the extended formulation from the Lagrangean dual.

Figure 2: Telecommunication traffic.

Figure 3: Feasible assignment of traffic on rings network (capacity = 60).

We assume that $d_e < C$ since otherwise part of the traffic on edge e is shared between $\lfloor \frac{d_e}{C} \rfloor$ rings devoted to edge e and the remaining traffic $d_e - C \lfloor \frac{d_e}{C} \rfloor$ is treated separately. We define L and K as respectively lower and upper bounds on the number of rings required in an optimal solution, with

$$L \geq \lceil \frac{\sum_{e \in E} d_e}{C} \rceil, \; K \leq |E|, \text{and } L \leq K. \tag{5}$$

Letting $x_e^k = 1$ if edge e is assigned on ring k and zero otherwise, and $y_i^k = 1$ if a multiplexer is installed at node i on ring k and zero otherwise, leads to a natural integer programming formulation which we refer to as $[F_{ND}]$.

$$Z = \min \quad \sum_k \sum_i y_i^k \tag{6}$$

$[F_{ND}]$ s.t.

Assign every edge	$\sum_k x_e^k$	$=$	1	$\forall e \in E$ (7)
Install multiplexer if needed	y_i^k	$\geq$	x_e^k	$\forall e, i, k$ with $e \in \delta(i)$ (8)
Capacity constraints	$\sum_e d_e x_e^k$	$\leq$	C	$\forall k$ (9)
Binary variables	$x_e^k, \; y_i^k$	$\in$	$\{0,1\}$	$\forall e, i, k$ (10)

where $\delta(i)$ is the set of edges incident to node i.

Let us show that this problem is NP-hard in the strong sense [20], because it contains the bin packing problem as a special case. Given an instance of the bin packing problem, that is a finite set E of items, a weight $d_e \in I\!N$ for each $e \in E$, a positive integer bin capacity C, we can define a corresponding instance for the telecommunication network design problem by letting the set of edges with their associated weight be equal to the set of items, and by specifying that all edges share a common node, node 0, while their other end point is not incident to any other edge (Figure 4 represents an instance with $|E| = 4$). Then in any solution of the network design problem, the total number of multiplexers is $|E| + B$ where B is the number of times

node 0 is duplicated. Equivalently, B is the number of bins used in the corresponding bin packing solution.

Figure 4: Restriction of the network design problem to a bin packing ($|E| = 4$).

The approximation offered by the linear relaxation of the integer program $[F_{ND}]$ is poor. Indeed if we take $L = \lceil \frac{\sum_{e \in E} d_e}{C} \rceil$, $x_e^k = y_i^k = \frac{1}{L}$ for all i, e, $k = 1, ..., L$, and zero otherwise, we have a feasible LP solution of cost $|V|$ which is a trivial lower bound on the objective value, assuming positive demands at each node.

Moreover, this problem has built in symmetry: all rings have identical characteristics. Formulation $[F_{ND}]$ allows alternate variable settings to represent the same physical solution. Variable fixing will often result in an alternate form of the same solution without forcing the objective value to rise.

Consequently, it is intractable to solve problems of practical size with a standard branch-and-bound approach whose bounds are based on the LP relaxation of $[F_{ND}]$ and whose branching scheme consist of setting x variables to zero or one (note that if all x's are integer in $[F_{ND}]$, then the y's are also integer). Instead, the reformulation resulting from decomposition leads to a much better linear relaxation bound, and takes advantage of the fact that all rings are identical.

In the rest of this Section, we derive the master formulation by means of Lagrangean duality. This shows how the master LP formulation and the Lagrangean dual are related (they are dual problems). We have found it interesting to emphasize the link between Lagrangean duality and Dantzig-Wolfe decomposition since it is typically exploited to produce lower bounds at every iteration of the column generation procedure. Indeed, each subproblem solution gives rise to a new (although not necessarily better) Lagrangean lower bound.

Observing $[F_{ND}]$, we note that, without the presence of equations (7), the problem decomposes into one for each ring. We proceed by performing a Lagrangean relaxation of the linking constraints. We associate, with each dualized linking constraint,

a weight π_e, called a Lagrange multiplier or dual variable. For any vector π, we define the Lagrangean function $LG(\pi)$ as

$$\begin{array}{rcll} LG(\pi) & = & \min & \sum_k \sum_i y_i^k + \sum_e \pi_e (1 - \sum_k x_e^k) \\ & & \text{s.t.} & \\ y_i^k & \geq & x_e^k & \forall e,\, i,\, k \text{ with } e \in \delta(i) \\ \sum_e d_e x_e^k & \leq & C & \forall\, k \\ x_e^k,\; y_i^k & \in & \{0,1\} & \forall\, e,\, i,\, k \end{array} \tag{11}$$

Noting that each ring has identical characteristics, we define a subproblem associated with a single ring:

$$\begin{array}{rcll} v(\pi) & = & \min & \sum_i y_i - \sum_e \pi_e\, x_e \\ [SP_{ND}] & & \text{s.t.} & \\ y_i & \geq & x_e & \forall e,\, i, \text{ with } e \in \delta(i) \\ \sum_e d_e x_e & \leq & C & \\ x_e,\; y_i & \in & \{0,1\} & \forall\, e,\, i \end{array} \tag{12}$$

which is a feasible problem whose optimal value is non-positive.

The Lagrangean function decomposes into the solution of K independent subproblems of this type:

$$LG(\pi) \;=\; \sum_e \pi_e + \sum_{k=1}^{K} v(\pi) \;=\; \sum_e \pi_e + K \times v(\pi) \tag{13}$$

As $LG(\pi)$ defines a lower bound on Z, we obtain the best possible such lower bound by maximizing over the choice of weight vector π.

$$Z_{LD} \;=\; \max_\pi \quad LG(\pi) \tag{14}$$

The above problem is called the Lagrangean dual. As we mentioned in a previous Section of this Chapter, it is the dual of the linear relaxation of the master problem that would result from applying Dantzig-Wolfe decomposition to the telecommunication network design problem formulation $[F_{ND}]$. Let us show how it relates to the master.

Since problem $[SP_{ND}]$ admits a finite set of feasible solutions, we can potentially proceed to enumerate them. Let $\{(c_q,\, a_q,\, r_q) \in I\!N \times \{0,1\}^{|E|} \times \{0,1\}^{|V|}\}_{q \in Q}$ be a

complete enumeration of the $|Q|$ undominated feasible solutions of $[SP_{ND}]$: $a_q = x$ for some $x \in \{0,1\}^{|E|}$ such that $\sum_e d_e x_e \leq C$,

$$c_q \;=\; \min_y \{ \sum_i y_i \;:\; y \in \{0,1\}^{|V|},\; y_i \geq a_{e\,q} \;\; \forall\, i,\, e \text{ with } e \in \delta(i)\}$$

and $r_q = y^*$ the solution of the above minimization problem. We adopt the following notation to restate the above in a more compact form:

$$\begin{array}{rrcll} c_q \;=\; \min_y \{ \sum_i y_i \;:\; a_q & = & x \in \{0,1\}^{|E|}, & & \\ r_q & = & y \in \{0,1\}^{|V|}, & & \\ y_i & \geq & x_e & \forall\, i,\, e \text{ with } e \in \delta(i), & \\ \sum_e d_e x_e & \leq & C \;\} & & \end{array}$$

Note that enumerating over all feasible solution of $[SP_{ND}]$ (including the dominated ones that do not minimize $\sum_i y_i$ for a given x) is also valid.

Then we linearize the Lagrangean dual (14), using an auxiliary variable $\mu \leq 0$, which stands for $v(\pi) \leq 0$.

$$\begin{array}{rrcll} Z_{LD} \;=\; \max & & \sum_e \pi_e \;+\; K\,\mu & & \\ \text{s.t.} & & & & \\ \sum_e \pi_e\, a_{e\,q} \;+\; \mu & \leq & c_q & \forall q \in Q & (15) \\ \mu & \leq & 0 & & \end{array}$$

By linear programming strong duality, we obtain

$$\begin{array}{rrcl} Z_{LD} \;=\; \min & & \sum_q c_q\, \lambda_q & \\ \text{s.t.} & & & \\ \sum_q a_{e\,q}\, \lambda_q & = & 1 & \forall e \in E \\ \sum_q \lambda_q & \leq & K & \\ \lambda_q & \geq & 0 & \forall q \in Q \end{array}$$

where λ is a vector of variables associated with constraints (15).

The theory of Lagrangean duality tells us that Z_{LD} is a better bound than the one resulting from the linear programming relaxation of $[F_{ND}]$, when the subproblem does not have the integrality property. This is the case for $[SP_{ND}]$ since

$$\begin{array}{l} conv(\{(x,\, y) \in \{0,1\}^{|E|} \times \{0,1\}^{|V|} :\; x_e \leq y_i \;\forall e, i,\; \sum_e d_e x_e \leq C\}) \;\subseteq \\ \qquad \{(x,\, y) \in [0,1]^{|E|} \times [0,1]^{|V|} :\; x_e \leq y_i \;\forall e, i,\; \sum_e d_e x_e \leq C\} \end{array} ,$$

and, typically, the inclusion is strict.

The latter formulation of the Lagrangean dual can be viewed as the linear relaxation of an alternate formulation for problem (P_{ND}). The incidence vectors a_q represent feasible edges to ring assignment patterns of cost c_q. The variable λ_q is one if pattern q is selected, zero otherwise. The integer programming reformulation of (P_{ND}), which we refer to as the *master*, is

$$
\begin{array}{lrll}
 & Z = \min & \sum_q c_q \lambda_q & \\
[M_{ND}] & \text{s.t.} & & \\
 & \sum_q a_{e\,q} \lambda_q = 1 & & \forall e \in E \\
 & L \leq \sum_q \lambda_q \leq K & & \\
 & \lambda_q \in \{0,1\} & & \forall q \in Q
\end{array}
\tag{16}
$$

Without lost of generality, we have explicitly added the lower bound on the number of selected patterns. If we can derive the explicit bounds L and K on the number of rings in any optimal solution using combinatorial arguments, then the linear relaxation of $[M_{ND}]$ produces a lower bound on Z which is even tighter than Z_{LD}.

2.2 Network Design with Split Assignments

The second application we present here is an extension of the telecommunication network design problem in which edges must be assigned twice, half of the traffic going on one ring, the rest being assigned to another ring. The natural IP formulation of this problem is analogous to $[F_{ND}]$, but the right hand side of the partitioning constraints take a more general form. Let b_e be the number of rings to which traffic demand associated edge e must be assigned. If d_e is even, we divide it by 2 and we set $b_e = 2$. Otherwise, we define two edges, e' and e'', to replace edge e, with $d_{e'} = \lfloor \frac{d_e}{2} \rfloor$ and $d_{e''} = \lceil \frac{d_e}{2} \rceil$. Let $S = \{(e', e'') : e' \text{ and } e'' \text{are new edges defined to replace an odd demand edge}\}$. We set $b_{e'} = b_{e''} = 1$ for all $(e', e'') \in S$. Moreover, we add explicit disjunctive constraints

$$
x^k_{e'} + x^k_{e''} \leq 1 \text{ for all } k, \text{ and for all } (e', e'') \in S\,, \tag{17}
$$

stating that edges e' and e'' may not be assigned to the same ring. Let m represent the number of edges after this transformation. Using the same notation as before, a compact integer programming formulation of this problem is

$$
\begin{array}{lrcll}
 & \min & & \sum_k \sum_i y_i^k & \\
[F_{NDSA}] & \text{s.t.} & & & \\
 & \sum_k x_e^k & = & b_e & \forall e = 1, \ldots, m \\
 & x_{e'}^k + x_{e''}^k & \leq & 1 & \forall (e', e'') \in S,\ k \\
 & y_i^k & \geq & x_e^k & \forall e,\ i,\ k \text{ with } e \in \delta(i) \\
 & \sum_e d_e x_e^k & \leq & C & \forall\, k \\
 & x_e^k,\ y_i^k & \in & \{0,1\} & \forall\, e,\ i,\ k
\end{array}
\tag{18}
$$

The Master we obtain from reformulating this problem is

$$
\begin{array}{lrcll}
 & \min & & \sum_q c_q \lambda_q & \\
[M_{NDSA}] & \text{s.t.} & & & \\
 & \sum_q a_{e\,q} \lambda_q & = & b_e & \forall e = 1, \ldots, m \\
L \leq & \sum_q \lambda_q & \leq & K & \\
 & \lambda_q & \in & \{0,1,2\} & \forall q \in Q,
\end{array}
\tag{19}
$$

where a_q represents a feasible edge assignment pattern, i.e. $a_q \in \{0,1\}^m : \sum_e d_e a_{e\,q} \leq C,\ a_{e'\,q} + a_{e''\,q} \leq 1,\ \forall (e', e'') \in S$; c_q is the associated cost, i.e. $c_q = \min_y \{\sum_i y_i : y \in \{0,1\}^{|V|},\ y_i \geq a_{e\,q},\ \forall i,\ e \text{ with } e \in \delta(i)\}$; and λ_q stands for the number of times pattern q is selected in the solution.

2.3 The Clustering Problem

The third application we consider is a graph partitioning problem that arises in contexts such as VLSI production, compiler design, or ordering of the computations in the Finite Elements method (see de Souza (1993) [13] and Johnson et. al. (1993) [29]).

Given an undirected graph $G = (V, E)$, edge costs c_e for each edge $e \in E$, weights $d_i \geq 0$ for each vertex (node) $i \in V$, and a capacity C (with $d_i \leq C$ for all $i \in V$), we wish to find a partition of V into clusters such that the sum of the node weights in each cluster does not exceed C, and that minimizes the sum of the costs of edges between clusters. Note that an equivalent objective is to maximize the sum of the costs on edges within the clusters. Without lost of generality, we assume that we have lower and upper bounds, respectively L and K, on the number of clusters in

an optimal partition (with $L \geq \lceil \frac{\sum_i d_i}{C} \rceil$ and $K \leq |V|$). Alternatively L and K can be part of the problem data.

Letting $x_i^k = 1$ if node i is assigned to cluster k, zero otherwise , and $y_e^k = 1$ if edge e is within cluster k, zero otherwise, a compact IP formulation for the above problem is

$$
\begin{array}{lrcll}
 & \max & \displaystyle\sum_{k=1}^{K} \sum_{e \in E} c_e \, y_e^k & & \\
[F_{CLUST}] & \text{s.t.} & & & (20) \\
 & \sum_{k=1}^{K} x_i^k & = & 1 & \forall i \in V \\
 & y_e^k & \leq & x_i^k & \forall e,\ i,\ k \text{ with } e \in \delta(i) \\
 & y_e^k & \geq & x_i^k + x_j^k - 1 & \forall e = (i,j),\ k, \\
 & \sum_i d_i \, x_i^k & \leq & C & \forall\, k \\
 & \sum_i x_i^k & \geq & 1 & \forall\, k = 1, \ldots, L \\
 & x_i^k,\ y_e^k & \in & \{0,1\} & \forall\, i,\ e,\ k
\end{array}
$$

When all the edge costs are non-negative, this problem is known as the Min-Cut Clustering problem (MCC), and is also called *the graph partitioning problem.* It is an NP-hard problem as shown in (Johnson et. al. (1993) [29]).

Using arguments similar to those we have used for the network design application, one can show that formulation $[F_{CLUST}]$ has an extremely weak linear programming relaxation and does not exploit the inherent problem symmetry. Here again a disaggregated formulation obtained by applying the decomposition principle can be derived. In [29] Johnson et. al. (1993) give an explicit comparison of the two formulations for this problem.

The set partitioning formulation that results from decomposition assumes that one can generate feasible clusters. Let $\{(c_q,\, a_q,\, r_q)\}_{q \in Q}$ be an enumeration of all feasible clusters, i.e. $a_q \in \{0,1\}^{|V|} : \ \sum_i d_i a_{i\,q} \leq C$, $c_q = \max_y \{ \sum_e c_e y_e \ : \ y_e \leq a_{i\,q} \ \forall i,\ e$ with $e \in \delta(i)$, and $y_e \geq a_{i\,q} + a_{j\,q} \ \forall e = (i,j)\}$, and $r_q = y^*$, a solution of this maximization problem. In our notation,

$$
\begin{array}{rcll}
c_q \ = \ \max_y \{ \sum_e c_e y_e \ : \ a_q & = & x \in \{0,1\}^{|V|}, & \\
r_q & = & y \in \{0,1\}^{|E|}, & \\
y_e & \leq & x_i & \forall\, i,\ e \text{ with } e \in \delta(i), \\
y_e & \geq & x_i + x_j - 1 & \forall e = (i,j),
\end{array}
$$

$$\sum_i d_i x_i \;\leq\; C \qquad \}.$$

Let $\lambda_q \in \{0,1\}$ for $q \in Q$ represent the decision of selecting cluster q in the solution. Then the master formulation is

$$\begin{array}{lrcll} & \max & \sum_q c_q \lambda_q & & \\ [M_{CLUST}] & \text{s.t.} & & & (21) \\ & \sum_q a_{i\,q} \lambda_q & = & 1 & \forall i \in V \\ L \;\leq & \sum_q \lambda_q & \leq & K & \\ & \lambda_q & \in & \{0,1\} & \forall q \in Q. \end{array}$$

2.4 The Single-machine Multi-item Lot-sizing Problem

The fourth application we consider differs from the previous ones in that the subsystems have different characteristics. The problem of finding a production plan for different goods (items) that share a common production facility (machine) has different variations. Here, we address the case where the machine has a limited production capacity, at most one item can be produced in each period and the objective is to minimize production, storage, set-up, and start-up costs.

Given demands d_t^i, the setup, startup, production, and storage costs, respectively c_t^i, f_t^i, p_t^i, and h_t^i, and the machine capacity U_t^i, for item $i = 1, \ldots, I$ over a discrete time horizon $t = 1, \ldots, T$ and given that the single machine can produce at most one type of item in each period (without lost of generality, we can assume that the single machine processes exactly one item in each period), the problem is to find a feasible production plan (i.e., a machine/period to item assignment with corresponding production level to satisfy demands while respecting the machine capacity constraint) that minimizes the overall cost.

To formulate this problem, we introduce the following variables:

$x_t^i = 1$ if the machine is setup for item i in period t, 0 otherwise;
$y_t^i = 1$ if the production of item i is started in period t, 0 otherwise;
$z_t^i =$ the production level of item i in period t;
$s_t^i =$ the stock level of item i at the beginning of period t.

A natural model for this fourth application is:

$$\min \;=\; \sum_{i=1}^{I} \sum_{t=1}^{T} (c_t^i\, x_t^i \;+\; f_t^i\, y_t^i \;+\; p_t^i\, z_t^i \;+\; h_t^i\, s_t^i)$$

$$
\begin{array}{lrcll}
[F_{MILS}] & \text{s.t.} & & & (22)\\
& \sum_i x_t^i & = & 1 & \forall t,\\
& z_t^i \;+\; s_t^i & = & d_t^i \;+\; s_{t+1}^i & \forall i,\ t,\\
& z_t^i & \leq & U_t^i\, x_t^i & \forall i,\ t,\\
& y_t^i & \geq & x_t^i \;-\; x_{t-1}^i & \forall i,\ t,\\
& x_t^i,\ y_t^i & \epsilon & \{0,1\} & \forall i,\ t,\\
& z_t^i,\ s_t^i & \geq & 0 & \forall i,\ t,
\end{array}
$$

After dualization of the partitioning constraints, the problem decomposes into one subproblem for each item. The subproblems are feasible if F_{MILS} is feasible, and they have a finite number of feasible solutions. Let $\{(c_q,\, a_q,\, r_q)\}_{q\in Q^i}$ be a complete enumeration of all feasible single item production plans for item i; i.e. $q \in Q^i$ iff

$$
\begin{array}{ll}
c_q = \min_{(y^i,\, z^i,\, s^i)} \{ & \sum_{t=1}^T (c_t^i\, x_t^i \;+\; f_t^i\, y_t^i \;+\; p_t^i\, z_t^i \;+\; h_t^i\, s_t^i) :\\
& a_q = (x_t^i, e^i) \in \{0,1\}^T \times \{0,1\}^I,\\
& \text{where vector } e^i \text{ has an entry 1 in row } i \text{ and zero's elsewhere,}\\
& r_q = (y^i,\, z^i,\, s^i) \in \{0,1\}^T \times I\!R_+^T \times I\!R_+^T,\\
& z_t^i \;+\; s_t^i \;=\; d_t^i \;+\; s_{t+1}^i \qquad \forall t,\\
& z_t^i \;\leq\; U_t^i\, x_t^i \qquad \forall t,\\
& y_t^i \;\geq\; x_t^i \;-\; x_{t-1}^i \qquad \forall t\ \}.
\end{array}
$$

Letting $Q = \cup_{i=1}^I Q^i$, and letting $\lambda_q \in \{0,1\}\ \forall q \in Q$ represent the decision of selecting production plan q in the solution, we formulate the Master as

$$
\begin{array}{lrcll}
 & \min & & \sum_q c_q\, \lambda_q & \\
[M_{MILS}] & \text{s.t.} & & & (23)\\
& \sum_q a_{t\,q}\, \lambda_q & = & 1 \qquad \forall t = 1,\ldots,T & (24)\\
& \sum_q a_{(T+i)\,q}\, \lambda_q & = & 1 \qquad \forall i = 1,\ldots,I & (25)\\
& \lambda_q & \in & \{0,1\} \qquad \forall q \in Q, &
\end{array}
$$

where we indexed by t (resp. $(T+i)$) the first T (resp. following I) components of vector a_q. Constraint (24) ensures that the single machine processes one type of item in each period, while constraint (25) accounts for the fact that there must be one complete production plan for each item. Note that the latter constraints imply that $\sum_q \lambda_q = |I|$.

3 IP Column Generation for Set Partitioning Problems

In this Chapter, we present a column generation based algorithm to solve exactly integer programs involving an enormous number of variables. We first give a column generation formulation of an integer program which we refer to as the master. We then articulate our theoretical presentation around this particular formulation.

The model $[M]$ that we propose below is a slightly generalized version of the set partitioning problem in which the right hand sides and the decision variables are positive integers not restricted to 0-1. This model is general enough to include all four applications presented in Chapter 2. The master integer program $[M]$ can either be the result of the reformulation of an IP by decomposition, or be the initial way to model the problem on hand.

The integer programming (IP) column generation algorithm we describe is an exact optimization procedure that integrates column generation with branch-and-bound. The bounding scheme used to prune the branch-and-bound tree is based on the linear relaxation of the master. Due to the large number of columns in the master, a column generation algorithm is used at each node of the branch-and-bound tree to solve the master linear relaxation. The branching scheme is made compatible with the column generation algorithm by modifying the subproblems appropriately.

After the model presentation, we successively describe the components of the IP column generation algorithm. First we present the procedure to solve the master LP at a branch-and-bound node. Then, we discuss the enumeration scheme to solve the master IP and we describe in more detail the branching scheme and the bounding scheme.

3.1 A Model for Generalized Set Partitioning

Let I be a set of m elements. We consider a set of subsets of I, the set of *feasible* subsets. With a feasible subset $I^q \subseteq I$, we associate an incidence vector $a_q \in \{0,1\}^m$, with $a_{i\,q} = 1$ if $i \in I^q$, $a_{i\,q} = 0$ otherwise, and a cost c_q. We let r_q represent the remaining characteristics of I^q (optional). In the model we consider, the set of feasible subsets can be described implicitly by a mixed integer program, i.e., $c(x) = \min_{(y,z)}\{fx + gy + hz,\ (x,y,z) \in W,\ x \in \{0,1\}^m,\ y \in I\!N^p,\ z \in I\!R^r_+\}$,

where x represent decision variables defining the subset, y and z represent auxiliary variables, f, g, and h represent the associated cost vectors, and W is a polyhedron that can be described by linear inequalities. In Chapter 1, we used similar notations, letting $S = W \cap (\{0,1\}^m \times I\!N^p \times I\!R^r_+)$. Any feasible vector x of the above optimization subproblem defines a feasible subset I^q represented by $c_q = c(x)$, $a_q = x$, and $r_q = (y^*, z^*) = argmin_{(y,z)}\{fx + gy + hz,\ (x,y,z) \in S\}$. Let $\{(c_q, a_q, r_q)\}_{q \in Q}$ be a complete list of all feasible subsets. Typically, the size of Q is large, the number of feasible subsets growing exponentially with m. In general, the problem is well defined if one is able to enumerate all the feasible subsets either implicitly or explicitly.

Given I, W, and $b_i \in I\!N\ \forall i$, the problem consists of finding a minimum cost selection of subsets that covers each element i exactly b_i times. Letting λ_q equal the number of times subset q is selected, we obtain an integer programming formulation:

$$
\begin{aligned}
& & Z \;=\; \min & \quad \sum_{q \in Q} c_q\, \lambda_q & & (26)\\
& [\mathrm{M}] & \text{s.t.} & & & \\
& \sum_{q \in Q} a_{i\,q}\, \lambda_q & = & \; b_i & \forall i \in I & (27)\\
& \sum_{q \in Q} \lambda_q & \leq & \; K & & (28)\\
& \sum_{q \in Q} \lambda_q & \geq & \; L & & (29)\\
& \lambda_q & \in & \; I\!N & \forall q \in Q & (30)
\end{aligned}
$$

Without lost of generality , we have added constraints (28) and (29) providing lower and upper bounds on the total number of selected subsets. Equations (27) represent the partitioning constraints. The standard set partitioning problem assumes that all b_i are 1 and that the λ's are restricted to be zero or one.

It is straightforward to see how this model includes the applications formulated in Chapter 2. Note that model $[M]$ even allows us to treat problems with non-identical subsystems such as the single-machine multi-item lot-sizing application. Numerous other applications can be modeled by $[M]$. To cite only one, we mention the single depot capacitate vehicle routing problem. In this case, I represents a set of clients to which one must deliver goods from a depot, and $b_i = 1\ \forall i$. A subset of clients is feasible if the total demand of these clients does not exceed the capacity of a vehicle. The associated cost is the length of the shortest route starting from the depot, visiting these clients and ending at the depot. One can generalize model $[M]$

further by letting the entries $a_{i\,q}$ be positive integers. The model would then include the cutting stock problem presented in Chapter 1.

3.2 Using Column Generation to Solve the Master LP

If we drop the integer restrictions in Model $[M]$, the resulting linear program $[M_{LP}]$ is not trivial to solve. It has typically a very large number of variables, each of which is associated with some feasible subset I^q (i.e. a feasible solution in S). So, it is practically impossible to generate the entire model. Instead, the master LP is solved by generating the columns if and when the LP solution method needs them, a technique that is called *column generation.*

The column generation procedure starts with a known basic feasible solution of the linear relaxation of the master $[M_{LP}]$. This initial solution can be generated by solving the corresponding phase 1 problem, i.e., initiating the formulation with appropriate artificial basic variables and using the column generation algorithm to solve this augmented problem with an artificial objective function penalizing the presence of artificial variables in the basis. Alternatively, one can generate a feasible solution heuristically or combine the phase 1 problem with the phase 2 problem as we shall see in the next Chapter.

Initially, the formulation of the linear program contains only a subset of the possible columns $\tilde{Q} \subseteq Q$, i.e. the basic feasible solution plus eventually a few other columns. We refer to it as the *restricted* master LP, $M_{LP}(\tilde{Q})$. Additional columns are generated as described below.

The algorithm proceeds by pricing out the variables (columns) that are not currently in the formulation, in order to check the optimality of the current basis. Most LP solvers produce the dual solution (the shadow prices) as a by-product of the primal optimization. So after solving the restricted master LP, $M_{LP}(\tilde{Q})$, the values of the dual variables are collected. Let $(\pi,\, \mu,\, \nu) \in I\!R^m \times I\!R^1_- \times I\!R^1_+$ be the dual variables associated with, respectively, the partitioning constraints (27) and the upper and lower bound cardinality constraints (28-29).

If the primal optimality condition holds, i.e., the minimum reduced cost is non-

negative

$$\min\{c_q - \sum_{i=1}^{m} \pi_i \, a_{i\,q} - \mu - \nu : \; q \in Q\} \geq 0, \tag{31}$$

the master LP problem has been solved without specifying all the columns or solving the full formulation. If not, the algorithm produces one (or more) non-basic columns q that violates dual feasibility, i.e., for which $\sum_{i=1}^{m} \pi_i \, a_{i\,q} + \mu + \nu > c_q$. Then the simplex method, when applied to the master LP, would introduce variable λ_q in the basis to improve the current solution, barring degeneracy. The column generation procedure accounts for this possibility by adding variable λ_q as a new column to the restricted master LP, augmenting $\tilde{Q}$. The new restricted formulation can be solved, using any LP solver, and the entire procedure can be repeated.

Observe that (31) is itself an optimization problem referred to as the pricing subproblem. Since the set Q is only known implicitly (cfr Section 3.1), the most negative reduced cost column, if any, is obtained by solving

$$\begin{aligned} v(\pi, \mu, \nu) \; = \; \min \{ \quad & (f - \pi)\, x \; + \; g\, y \; + \; h\, z \; - \; \mu \; - \; \nu : \\ & (x,\, y,\, z) \in W, \\ & x \in \{0,1\}^m, \\ & y \in I\!N^p, \\ & z \in I\!R^r_+ \quad \} \end{aligned} \tag{32}$$

whose solution (x^*, y^*, z^*) defines a new column $a_q = x^*$, $r_q = (y^*, z^*)$ of cost $c_q = f\, x^* \; + \; g\, y^* \; + \; h\, z^*$.

To be effective, the column generation algorithm requires that the subproblem can be solved efficiently, and that an optimal solution to the master LP be obtained before too many columns have been added to the formulation of the restricted master LP.

3.3 Solving the Master IP

Unless the solution of the master LP is integer, solving $[M_{LP}]$ does not provide a solution to the original IP problem $[M]$. This means that one has to turn to branch-and-bound in order to find an optimal IP solution. Applying a standard branch-and-bound algorithm to the restricted master is not an exact optimization procedure since there is no guarantee that an optimal solution (or even a feasible

IP solution) can be found from among the existing columns. However, an optimal solution of the restricted master integer program provides an heuristic solution (i.e. the best integer solution that can be obtained by combining existing columns).

Note that, when the standard branch-and-bound algorithm is applied to the restricted master, the LP relaxation bounds are valid lower bounds for the restricted master only. At a given node, in order to obtain a valid lower bound for the unrestricted node master problem, one must prove LP optimality. That is one must solve the pricing subproblem since after branching some new columns might price out negatively. Using column generation at the node leads to a valid lower bound for the unrestricted node master problem and generates additional columns that might be part of the optimal integer solution to the unrestricted master.

Combining column generation with branch-and-bound, one can solve problem $[M]$ optimally. At each node u of the branch-and-bound enumeration tree, the above column generation algorithm is applied to solve the linear relaxation of the master $[M^u_{LP}]$. At each iteration of the column generation procedure at node u, the restricted master LP $[M^u_{LP}(\tilde{Q})]$ is solved giving an upper bound $Z^u_{LP}(\tilde{Q})$ on the value of the linear programming relaxation $Z^u_{LP}(Q)$. The process continues until LP optimality is proved. Then $Z^u_{LP}(Q) = Z^u_{LP}(\tilde{Q})$. Next, either node u is pruned by bound, IP optimality, or infeasibility; or the optimal LP solution is fractional. In the latter case, we branch; that is, we separate the feasible solution set of the node problem into more restricted subsets whose union does not contain the current fractional solution, but contains all integer solutions.

Embedding column generation in integer programming techniques raises some specific difficulties. First, a branching scheme compatible with column generation must be derived since applying a conventional integer programming branching scheme is not straightforward nor recommended in this context. Second, the efficiency of branch-and-bound algorithm will suffer from the tailing-off effect of the column generation algorithm (i.e., the large number of iterations needed to prove LP optimality). But is it really necessary to solve the master LP to optimality in order to derive a bound at the node? In the next two sections we address these issues.

3.4 Branching

Solving integer programs using a branch-and-bound algorithm based on the LP relaxation assumes that all fractional solutions can be eliminated by successive separation of the feasible solution space. A branching scheme is a set of rules that enables one to exclude any given fractional solution while making sure that the resulting separation of the IP feasible solution space is valid. In mathematical terms, let X^{u_0} represent the set of LP feasible solutions at node u_0, given λ^* a current fractional feasible solution of $[M_{LP}^{u_0}]$, the branching scheme must provide a way to separate X^{u_0} into $X^{u_1}, \ldots, X^{u_p}$ such that $\lambda^* \notin (\cup_{i=1}^{p} X^{u_i})$ and $(X^{u_0} \cap I\!N^{|Q|}) \subseteq (\cup_{i=1}^{p} X^{u_i})$. Moreover, to guarantee final termination of the branch-and-bound algorithm, it must be proved that, after a finite number of separations, the solutions of the leaf node master problems are integer. In this thesis, we will use only a binary enumeration trees (i.e. $p = 2$); so, after separation, node u_0 has two successor nodes, u_1 and u_2.

A *good* branching scheme is one that, in addition to having the properties presented above, takes into consideration the performance of the branch-and-bound algorithm. This is a qualitative concept. Recall that branch-and-bound is a procedure to implicitly enumerate all integer solutions. It works on two fronts, on the one hand trying to construct an optimal integer solution and on the other trying to produce a tighter lower bound that proves the optimality of the current integer solution. So, among alternate branching schemes leading to a valid separation, one wishes to select the one that maximizes the value of the minimum lower bound over all the successor nodes, and that improves one's chance of generating intermediate integer solutions. In the operations research literature, one can find many heuristic arguments to qualify a *good* branching scheme. Such rules of thumb are: a separation leading to a partition of the feasible space (s.t. $X^{u_1} \cap X^{u_2} = \emptyset$) is better than a covering; one should try to split the solution space into subspaces of approximately equal size; one should try to eliminate as many fractional solutions as possible in one separation.

In the column generation context, other considerations must be taken into account as well. First, we need to show that the branching scheme we use is compatible with the column generation algorithm. That is, we can still use column generation to solve the LP relaxation of the more restricted problems defined at successor nodes. Moreover, when it comes to qualify a *good* branching scheme, we insist on keeping the master and the subproblem tractable.

In the rest of this section, we show how separation can be implemented in a column generation framework. We present an earlier approach of Ryan and Foster [1981]. Then we propose to extend Ryan and Foster branching scheme to the case of model $[M]$. We next prove the validity of our branching scheme. Finally we present a generalization of this branching rule for the case where the matrix entries in model $[M]$ are general integers not restricted to zero or one. To begin, we show why the conventional branching rule based on variable dichotomy is not appropriate.

3.4.1 Conventional branching

To motivate our assertion, let us look at the fractional solution obtained for the small instance of the telecommunication network application given in Chapter 2 (see Figure 5). A feasible integer solution for this problem is given in figure 6; its cost is 7. In table 1, we present a few feasible patterns (columns) for this problem. The edges have been numbered in increasing order of the associated traffic demands (i.e. edge $1 = (A, B), \ldots,$ edge $6 = (C, D)$).

Figure 5: An instance of the network design (ND) problem.

Figure 6: A feasible solution (with capacity = 60).

At the root node of the master branch-and-bound tree (node 0), $\lambda_2 = \lambda_3 = \lambda_4 = \lambda_5 = \frac{1}{2}$ is an optimal solution of $[M^0_{LP}(Q)]$. The corresponding value $Z^0_{LP}(Q)$ is 6. Conventional branching consists of selecting a fractional variable, say λ_2, and imposing $\lambda_2 \leq 0$ on one branch, and $\lambda_2 \geq 1$ on the other. In figure 7, we present the resulting branch-and-bound tree where for each node we give the upper and lower

e	d_e	1	2	3	4	5	6	7	8	9	10	11	12
1	10	1	0	1	0	1	0	1	1	1	0	0	1
2	12	1	0	0	1	1	1	1	0	1	0	0	1
3	14	1	0	1	1	0	1	0	1	1	1	0	0
4	16	0	1	0	0	1	1	0	1	1	0	0	1
5	18	0	1	1	0	0	1	1	0	0	0	0	0
6	20	0	1	0	1	0	0	1	1	0	0	1	1
	cost	4	3	3	3	3	4	4	4	4	2	2	4

Table 1: Some feasible patterns for the network design instance.

bound on the node problem value.

Figure 7: Branching on λ_2.

At node 2, edges 4, 5 and 6 are covered by pattern 2, the remaining problem consists of finding an optimal assignment for edges 1, 2 and 3. Since these edges can be assigned together on the same ring, $\lambda_1 = \lambda_2 = 1$ is an optimal LP solution and the node can be pruned by optimality.

At node 1, $\lambda_2 \leq 0$ implies that pattern 2 cannot be used; so, we delete it from the formulation of the restricted master. But we must also make sure that it will not be regenerated. This is done by adding to the subproblem $[SP_{ND}]$ (cfr (12)), the constraint

$$x_1 + x_2 + x_3 + (1 - x_4) + (1 - x_5) + (1 - x_6) \geq 1. \tag{33}$$

After application of the column generation algorithm, we find a new fractional solution $\lambda_6 = \lambda_7 = \lambda_8 = \frac{1}{2}$ of cost 6. So the lower bound is no tighter than the bound at the root node. A new separation must take place.

In conclusion, we observe that the variable fixing branching scheme leads to an unbalanced separation. On the right branch, the feasible space is very restricted while on the left branch, only one pattern is excluded, leading to a problem that

is not much more restricted than the one at the predecessor node. Moreover, the constraint added to the subproblem is not trivial and it may destroy the structure of the subproblem, making column generation difficult by means of a heuristic or a dynamic program and maybe making exact column generation intractable as well.

3.4.2 Ryan and Foster branching scheme for standard set partitioning

For the standard set partitioning problem (with right hand side restricted to 1), Ryan and Foster (1981) [38] suggest a more appropriate branching strategy based on the following proposition.

Proposition 2 (Ryan and Foster, 1981) *If $b_i = 1 \ \forall i = 1\ldots,m$, and λ is a fractional solution of $[M_{LP}]$, then there exist e and $f \in \{1\ldots,m\}$ such that*

$$0 < \sum_{q:a_{eq}=a_{fq}=1} \lambda_q < 1. \tag{34}$$

Having identified such a pair of rows (e, f), one can cut off the current fractional solution and perform a valid separation by imposing $\sum_{q:a_{eq}=a_{fq}=1} \lambda_q \leq 0$ on the left branch and $\sum_{q:a_{eq}=a_{fq}=1} \lambda_q \geq 1$ on the right branch. Since the number of such pairs of distinct elements from $\{1\ldots,m\}$ is finite, the branch-and-bound algorithm will terminate with an integer solution.

This branching strategy is typically compatible with column generation. On the left branch, it amounts to eliminating all the columns with $a_{eq} = a_{fq} = 1$ from the master formulation and, to avoid to regenerate such columns, we require that all newly generated patterns (i.e. subproblem solutions) satisfy $a_{eq} + a_{fq} \leq 1$. This latter requirement amounts to adding the constraint $x_e + x_f \leq 1$ to the subproblem formulation. On the right branch, $\sum_{q:a_{eq}=a_{fq}=1} \lambda_q \geq 1$ implies that elements e and f will be covered exclusively with patterns that include them both; meaning that patterns containing only one of them are excluded from the solution, i.e. $\sum_{q:a_{eq}\neq a_{fq}} \lambda_q \leq 0$. So we eliminate all columns with $a_{eq} \neq a_{fq}$ from the master formulation, and we require that all new patterns satisfy $a_{eq} = a_{fq}$, that is, we add the constraint $x_e = x_f$ in the subproblem.

Let us illustrate this branching strategy on the network design application instance. Given the fractional solution at the root node, we note that

$$w_{56} = \sum_{q:a_{5q}=a_{6q}=1} \lambda_q = \frac{1}{2}, \tag{35}$$

where we have introduced w_{56} as an auxiliary variable to represent the co-assignment of edges 5 and 6. In an integer solution, edges 5 and 6 are assigned to either the same ring ($w_{56} = 1$) or different rings ($w_{56} = 0$). So the branching scheme proposed by Ryan and Foster can be interpreted as a conventional branching rule on such auxiliary variables.

Figure 8: Branching on w_{56}.

We separate by letting $w_{56} \leq 0$ on the left branch and $w_{56} \geq 1$ on the right branch (see figure 8). After reoptimization of the successor node master LP, we obtain the fractional solutions $\lambda_2 = \lambda_7 = \lambda_9 = \lambda_{10} = \frac{1}{2}$ with cost 6.5 at node 1 (see Table 1), and $\lambda_3 = \lambda_6 = \lambda_{11} = \lambda_{12} = \frac{1}{2}$ with cost 6.5 at node 2. Both nodes can be pruned because we know an integer solution of cost 7 (see Figure 6), all integer solutions have integer cost and $\lceil 6.5 \rceil \geq 7$.

In conclusion, with Ryan and Foster branching scheme, one separates the solution space into subsets of approximately equal size, and the constraints one has to add to the pricing subproblem are quite simple: constraints like $x_e = x_f$ amount to defining groups of elements; while $x_e + x_f \leq 1$ are disjunctive constraints. As we shall see in the next Chapter, for the applications we consider, these subproblem constraints can be incorporated in a heuristic column generation procedure and they do not very much increase the difficulty of solving the subproblem exactly. The co-assignment ($x_e = x_f$) and disjunctive ($x_e + x_f \leq 1$) constraints that result from branching progressively induce a partition of the column set into subclasses, each of which is covering a subset of rows. Consequently, as expected from a branching scheme, the intermediate master LP solutions are more likely to be integral.

There are many ways of implementing Ryan and Foster branching rule. The selection of rows on which to branch can be extended to rows that are not explicitly in the formulation $[M]$, but that could be added to define auxiliary variables, i.e., rows associated with the components of r_q (cfr Section 3.1). For instance, in the network design application, letting $r_{iq} = 1$ if a multiplexer is installed at node i in pattern q, we can add to $[M_{ND}]$ rows such as $w_i = \sum_{q \in Q: r_{iq}=1} \lambda_q$, where w_i represent the total number of multiplexers installed at node i. Then, we may branch on a pair of rows one of which corresponds to an edge and the other to a node (note that the extra rows do not have to be introduced explicitly in order to use this type of branching).

In some applications, one can show that all fractional solutions can be eliminated using only a subset of pairs of rows. This is typically the case when there is a one to one correspondence between the integer variables in the original formulation $[F]$ and auxiliary variables in the reformulation $[M]$, and when these auxiliary variables can be defined as a subset sum of λ's such as (34). For instance, in the multi-item lot-sizing application (22-23), it is enough to enforce the integrality of the machine setup variables ($x_t^i = 1$ if the machine is setup for item i in period t, zero otherwise) to obtain an integer solution of $[F_{MILS}]$. Moreover, there is a one to one correspondence between these original integer variables and particular subset sums of λ's: $x_t^i = \sum_{q: a_{tq} = a_{(T+i)q}=1} \lambda_q$. If $\sum_{q: a_{tq} = a_{(T+i)q}=1} \lambda_q$ is integer for all $\{t, i\}$, then the corresponding solution x in $[F_{MILS}]$ is integer. Conversely, if all x_t^i are integer, then all λ's are integer in $[M_{MILS}]$ (as shown in [4]). So, if all subsets of 2 rows, one of which corresponds to a period and the other to an item, lead to an integer subset sum of λ's, then λ is integer.

3.4.3 A branching scheme for model $[M]$

When the right hand sides in $[M]$ are not restricted to 1, the previous branching scheme is not sufficient to eliminate all fractional solutions. For instance, consider a problem in which $m = 3$ and $b_i = 2 \; \forall i = 1, \ldots, m$. In Table 2, we have enumerated all possible patterns, assuming that they are all feasible. Assuming the current fractional solution is $\lambda_q = \frac{1}{2} \; \forall q$, it is not possible to eliminate this fractional solution based on the subset sum of λ's defined by Ryan and Foster (34). One can easily check that for any pair of rows (e, f), $\sum_{q: a_{eq} = a_{fq}=1} \lambda_q = 1$.

However, considering a third row, one can define a subset of patterns whose λ's sum is fractional: $\sum_{q: a_{1q} = a_{2q} = a_{3q}=1} \lambda_q = \lambda_1 = \frac{1}{2}$. The fractional solution can then

1	1	1	0	1	0	0
1	1	0	1	0	1	0
1	0	1	1	0	0	1

Table 2: Tableau with all possible patterns when $m = 3$ and $b_i = 2\ \forall i = 1, \ldots, m$.

be eliminated by enforcing that one must select an integer number of patterns from that subset. This example illustrates how Ryan and Foster branching scheme can be extended to include the case of general integer RHS. Next, we give an intuitive explanation of the way we eliminate a fractional solution of model $[M]$. Then, in Proposition 3, we formalize the result on which we base our branching scheme. The constructive proof paves the way for a practical separation scheme.

Branching for model $[M]$ is based on the identification of a subset of patterns whose variable sum is fractional. In the search for a fractional sum of λ's, one can ignore patterns whose λ is integer. As in the Ryan and Foster branching scheme, one tries to derive a subset of patterns that includes some fractional patterns and excludes others. To identify such a subset of pattern, one partitions the fractional λ's according to the elements covered by the associated patterns. A restriction to fractional patterns that contains a given element i leads to a sum of λ's bounded by b_i. The class of fractional patterns containing element i is then partitioned further by selecting another element that is included in some of the remaining patterns but not all. This leads to a new subclass of patterns whose sum of variables is either fractional or bounded above by an integer which is strictly smaller than the bound one had before partitioning. Pursuing this idea, we generalize the previous result of Ryan and Foster (Proposition 2) and we derive a branching scheme for model $[M]$.

Let us introduce some notation. Let $T \subseteq I = \{1, \ldots, m\}$ be represented by its incidence vector a_T ($a_{iT} = 1$ if $i \in T$, zero otherwise). Pattern q satisfies $a_q \geq a_T$ if $a_{iq} \geq a_{iT}\ \forall i = 1, \ldots, m$. Let $b_{\max} = \max\{b_i : i \in I\}$, and $|a_q|$ be the size of the element set represented by a_q: $|a_q| = \sum_{i \in I} a_{iq}$.

Proposition 3 *If λ is a fractional solution of $[M_{LP}(\tilde{Q})]$, then there exist a subset $T \subseteq I$ such that*

$$\sum_{q \in \tilde{Q}: a_q \geq a_T} \lambda_q \quad \textit{is fractional,} \tag{36}$$

and $|T| \leq \min(\min_i\{b_i + 1 : \exists q \textit{ with } a_{iq} = 1 \textit{ and } \lambda_q \textit{ is fractional}\}, \max_q\{|a_q| : \lambda_q \textit{ is fractional}\}) \leq b_{\max} + 1$.

Proof. To prove the result, we first show that the following claim is valid:
if there exists $S \subseteq I$ such that $\sum_{q:a_q \geq a_S} \lambda_q = n \in I\!N^1$ and λ_k is fractional for some k with $a_k \geq a_S$, then there exists $R \subseteq I \setminus S$ and $T = S \cup R$ such that $\sum_{q:a_q \geq a_T} \lambda_q$ is fractional and $|R| \leq n$.
We use an inductive argument on n. Assuming that this observation is true for $n = 1, \ldots, r$, let us prove that it is still true for $n = r + 1$. Assume that λ_k is fractional with $a_k \geq a_S$ and $\sum_{q:a_q \geq a_S} \lambda_q = r + 1$ is integer. Then, as $\sum_{q:a_q \geq a_S} \lambda_q$ is integer, there exists a second pattern $k' \in \tilde{Q}$ with $a_{k'} \geq a_S$ and $\lambda_{k'}$ is fractional. There are no duplicate columns in $\tilde{Q}$, so $\exists j \in I$ s.t. $a_{j\,k} \neq a_{j\,k'}$. Assume that $a_{j\,k} = 1$ and $a_{j\,k'} = 0$ (otherwise invert the roles of k and k'). Let $R = \{j\}$ and $T = S \cup R$. Then

$$0 < \sum_{q:a_q \geq a_T} \lambda_q \leq r + 1 - \lambda_{k'} < r + 1 \tag{37}$$

Thus, the subset sum defined by T is either fractional and we have found an appropriate set T, or it is integer and smaller than or equal to r. In the latter case, the induction hypothesis applies since for $S = T$, $a_k \geq a_S$ and λ_k is fractional. Using a similar argument it is straightforward to prove that the above claim is also true for $n = 1$.
Now, assuming that the current solution λ is fractional, we let $b_{i^*} = \min_i\{b_i : \exists q \text{ with } a_{i\,q} = 1 \text{ and } \lambda_q \text{ is fractional}\}$. For $S = \{i^*\}$, $\sum_{q:a_q \geq a_S} \lambda_q = b_{i^*} \in I\!N^1$. Applying the above claim leads to the conclusion that there exists T such that $|T| \leq b_{i^*} + 1$ and $\sum_{q:a_q \geq a_T} \lambda_q$ is fractional. Moreover $|T| \leq \max\{|a_q| : \lambda_q \text{ is fractional }\}$, since elements of T are chosen from the subset of elements covered by a fractional pattern. ∎

Separation

Given a set T leading to a fractional subset sum $\sum_{q:a_q \geq a_T} \lambda_q = \alpha \notin I\!N^1$, separation is performed by imposing

$$\sum_{q:a_q \geq a_T} \lambda_q \leq \lfloor \alpha \rfloor \tag{38}$$

on one branch, and

$$\sum_{q:a_q \geq a_T} \lambda_q \geq \lceil \alpha \rceil \tag{39}$$

on the other branch. We need to show that such a branching scheme leads to final termination of the branch-and-bound algorithm and can be implemented in a column generation framework.

We can guarantee that a feasible integer solution will be found (or infeasibility proved) after a finite number of branches. Indeed, the number of subsets T is finite,

and, since $b_{\max}$ is a valid upper bound on the value of a subset sum like (36), the number of dichotomies is finite as well. Moreover, Proposition 3 implies that if there is no set $T \subseteq I$ leading to a fractional subset sum like (36), then the solution of the master must be integer.

Given a fractional basic solution λ, the first step towards implementing this branching scheme is to find a fractional subset sum (36), while keeping $|T|$ as small as possible. The proof of proposition 3 tells us how to construct a subset T. Alternatively, one can enumerate all subsets of 2 rows before going on to subsets of 3 rows, etc.., until one identifies a fractional subset sum. This is a reasonable procedure when $b_{\max}$ is small or m (i.e. the maximum number of element in a column) is a small number. Then, we add the branching constraint (38) or (39) to the master formulation $[M^u_{LP}]$.

For a correct computation of the reduced cost of a column, one must then take into account the additional dual variables associated with the new branching constraint in the master. Consequently, along with the branching scheme, we present a pricing subproblem modification scheme. We define auxiliary variables for the subproblem. If a branching constraint concerns a subclass of pattern q such that $a_q \geq a_T$ for some $T \subseteq I$, we define an auxiliary variable $w_T \in \{0, 1\}$ in the subproblem such that $w_T = 1$ if the pattern defined by the subproblem solution includes subset T, and zero otherwise. Then w_T also appears in the objective of the pricing subproblem weighted by the appropriated shadow price of the master LP.

Note that the definition of the subproblem auxiliary variables can be enforced using linear inequalities. Given $T \subseteq I$, we add the following set of constraints to the subproblem to define w_T:

$$w_T \geq 1 - \textstyle\sum_{i \in T}(1 - x_i) \tag{40}$$

$$w_T \leq x_i \qquad \forall i \in T \tag{41}$$

The integrality of the x variables will guarantee the integrality of w_T with no need to specify explicitly that $w_T \in \{0, 1\}$. If the master branching constraint is of type (38) (resp. (39)), then the corresponding dual variable represents a penalty (resp. a reward) for selecting a pattern that includes T as a subproblem solution, and only constraint (40) is (resp. constraints (41) are) active in defining w_T.

Node master and subproblem formulation

We are now in a position to give an explicit formulation of the master problem and

the associated pricing subproblem at a given node u of the branch-and-bound tree.

Let G_0^u (resp. H_0^u) be the set of branching constraints of type (38) (resp. (39)) that have led to the definition of the problem at node u. That is, we let $g \in G_0^u$ if one of the branching constraints defining the problem at node u is

$$\sum_{q:a_q \geq a_g} \lambda_q \leq K^g \tag{42}$$

where a_g is the incidence vector of a specific set $T \subseteq I$ and $K^g \in I\!N^1$. Equivalently, $h \in H_0^u$ if $\sum_{q:a_q \geq a_h} \lambda_q \geq L^h$ is part of the definition of M^u.

Since the initial cardinality constraints (28-29) of model $[M]$ have the same form, we let a_0 be the null vector, $K^0 = K$, $L^0 = L$, $G^u = G_0^u \cup \{0\}$ and $H^u = H_0^u \cup \{0\}$. Then, the formulation of the master at node u is

$$\begin{aligned}
Z^u(Q) \;=\; \min \quad & \sum_{q \in Q} c_q \lambda_q & & \\
[M^u] \qquad \text{s.t.} & & & \qquad (43)\\
\sum_{q \in Q} a_{i\,q} \lambda_q \;&=\; b_i & \forall i \in I & \qquad (44)\\
\sum_{q \in Q: a_q \geq a_g} \lambda_q \;&\leq\; K^g & \forall g \in G^u & \\
\sum_{q \in Q: a_q \geq a_h} \lambda_q \;&\geq\; L^h & \forall h \in H^u & \\
\lambda_q \;&\in\; I\!N & \forall q \in Q &
\end{aligned}$$

We refer to its linear programming relaxation by $[M_{LP}^u(Q)]$ whose optimal objective value is $Z_{LP}^u(Q)$.

Let $(\pi,\, \mu,\, \nu) \in I\!R^m \times I\!R_-^{|G^u|} \times I\!R_+^{|H^u|}$ be an optimal dual solution of the linear relaxation of the restricted master at node u, $M_{LP}^u(\tilde{Q})$ with $\tilde{Q} \subseteq Q$, i.e.

$$Z_{LP}^u(\tilde{Q}) = \sum_{i \in I} \pi_i b_i + \sum_{g \in G^u} \mu_g K^g + \sum_{h \in H^u} \nu_h L^h. \tag{45}$$

Given the current set of shadow prices, the reduced cost of a column q is

$$\overline{c_q} = c_q - \sum_{i \in I} \pi_i a_{i\,q} - \sum_{g \in G^u: a_q \geq a_g} \mu_g - \sum_{h \in H^u: a_q \geq a_h} \nu_h. \tag{46}$$

So, the pricing subproblem, which aims at identifying a feasible column of minimum reduced cost, can be formulated as

$$v^u(\pi, \mu, \nu) = \min\{ \quad fx + gy + hz - \pi x - \textstyle\sum_{g \in G_0^u} \mu_g w_g - \sum_{h \in H_0^u} \nu_h w_h - \mu_0 - \nu_0$$

$$
\begin{aligned}
[SP^u] \qquad \text{s.t.} \quad & (x,\, y,\, z) \in W \cap \{0,1\}^m \times I\!N^p \times I\!R^r_+ && (47)\\
& w_k \geq 1 - \textstyle\sum_{i:a_{i\,k}=1}(1 - x_i) && \forall k \in G^u_0 \cap H^u_0\\
& w_k \leq x_i && \forall i : a_{i\,k} = 1,\ \forall k \in G^u_0 \cap H^u_0\ \}
\end{aligned}
$$

and the LP optimality condition (31) becomes

$$v^u(\pi,\, \mu,\, \nu) \geq 0. \tag{48}$$

When LP optimality does not hold, the solution of the subproblem defines a new column with $c_q = fx + gy + hz$ and $a_q = x$ that we add to the master formulation.

Simplifications for subclasses of branching constraints

The way the branching constraints are enforced in the master and the associated subproblem modifications simplify when K^g (resp. L^h) take its minimum (resp. maximum) value. Then, the master branching constraint amounts to eliminating columns from the formulation and the definition of the auxiliary variable in the pricing subproblem is replaced by an explicit constraint.

If $K^g = 0$, then $\sum_{q:a_q \geq a_g} \lambda_q = 0$ implies that no pattern with $a_q \geq a_g$ can be used in the solution; so, these columns are removed from $\tilde{Q}$ and the constraint

$$\sum_{i:a_{i\,g}>0} x_i \leq |a_g| - 1, \tag{49}$$

where $|a_g| = \sum_i a_{i\,g}$, is added to the subproblem.

Another special case arises when the lower bound L^h is also an upper bound on the subset sum defined by a_h. Note that, for any element i in the set defined by a_h, i.e., for all $i : a_{i\,h} = 1$,

$$\sum_{q:a_q \geq a_h} \lambda_q \leq \sum_{q:a_{i\,q}=1} \lambda_q = \sum_q a_{i\,q}\,\lambda_q = b_i \tag{50}$$

If, for some $h \in H^u$, $L^h = \min_i\{b_i : a_{i\,h} = 1\}$, then the elements in $S = \{i \in I : a_{i\,h} = 1 \text{ and } b_i = L^h\}$ must be covered exclusively with pattern such that $a_q \geq a_h$. Consequently, one can remove from the master formulation any column q such that $\sum_{i\in S} a_{i\,q} > 0$ and $a_{i\,q} < a_{i\,h}$ for some i. To avoid regenerating such patterns in the pricing subproblem, we constrain new columns that contain one element of S to cover all the other elements in the set defined by a_h, i.e.

$$x_i \geq x_j \text{ for all } i \in I : a_{i\,h} = 1 \text{ and for all } j \in S. \tag{51}$$

Note that any feasible solution of the master that uses none of the deleted patterns must satisfy $\sum_{q:a_q \geq a_h} \lambda_q = L^h$.

We point out that applying the general scheme in these special cases is valid but not as efficient. In the case $K^g = 0$, for instance, whatever the value μ_g of the dual variable associated with the branching constraint $\sum_{q:a_q \geq a_g} \lambda_q \leq 0$, $\mu_g{}' = \mu_g - \epsilon$ defines an alternate dual optimal solution $\forall \epsilon \in I\!R_+$ (since this new dual value is dual feasible for the restricted master LP and leads to the same objective value). So, if a column that includes the set defined by a_g has negative reduced cost (cfr (46)), there exists an alternate dual solution for which this column prices out non-negatively. Thus it is unnecessary to include this column in the formulation. In the case L^h is equal to an upper bound on the number of columns s.t. $a_q \geq a_h$, it can also been shown that there exists a dual solution for which the columns we explicitly excluded would price out non-negatively.

It is straightforward to check that the Ryan and Foster branching scheme is a specialization of the above branching scheme for the case $b_{\max} = 1$.

3.4.4 Generalization to the case where $[M]$ admits integer entries $a_{i\,q}$

In Chapter 1, we have introduced a model $[P]$ (see Section 1.3) which is more general than the model $[M]$ treated in this chapter, in the sense that the entries of the matrix, the $a_{i\,q}$'s, are general integers not restricted to 0-1. One can extend the above branching scheme to model $[P]$, as we shall see in this section. The following presentation is mainly theoretical since, to date, we have not applied this branching scheme to any practical problem. Although the proposed branching scheme seems to raise questions concerning the tractability of the modified subproblem, it may take a simpler form once adapted to a particular application.

Proposition 4 *Given a fractional solution λ of $[P]$, there exist $p \in I\!N^n$ such that*

$$\sum_{q:a_q \geq p} \lambda_q \quad \textit{is fractional}, \tag{52}$$

Proof. Take $p \in I\!N^n$ to be any maximal element of the set $S = \{a_q : q \in Q \text{ and } \lambda_q \text{ is fractional}\} \subseteq I\!N^n$. Such an element $p = a_{q^*} \in S$ always exists as Q

is a finite set. Then $\sum_{q:a_q \geq p} \lambda_q = \lambda_{q^*}$ is fractional. ∎

A branching scheme based on Proposition 4 eliminates a fractional solution by identifying a column p such that $\sum_{q:a_q \geq p} \lambda_q = \alpha \notin I\!N^1$ and by replacing the node master problem by two more restricted problems defined by the additional constraints $\sum_{q:a_q \geq p} \lambda_q \leq \lfloor \alpha \rfloor$ and $\sum_{q:a_q \geq p} \lambda_q \geq \lceil \alpha \rceil$ respectively. The proof of Proposition 4 mentions how to identify such a vector p. As Q is finite and $\sum_{q:a_q \geq p} \lambda_q \leq \sum_{i=1}^{m} b_i$ for all p, the number of such separations necessary to yield integer solutions is finite.

The additional constraints added to the master formulation in the course of the branching scheme are taken into account in the reduced cost computation by modifying the subproblem accordingly: We introduce auxiliary variables of the type $w = 1$ if $x \geq p$ and zero otherwise, which are added to the pricing subproblem objective function with coefficients equal to the dual variables associated with the targeted master branching constraints.

The correct setting of the auxiliary variable w may be enforced using a MIP formulation. Let $P^+ = \{i \in I : p_i > 0\}$. Introduce a binary variable $\eta_i \ \forall i \in P^+$ where $\eta_i = 1$ if $x_i \geq p_i$ and $\eta_i = 0$ if $x_i < p_i$. Then, to define w, the following system must be added to the pricing subproblem formulation:

$$p_i \eta_i \leq x_i \leq (p_i - 1) + (b_i - p_i + 1)\eta_i \quad i \in P^+$$

$$w \leq \eta_i \quad i \in P^+$$

$$w \geq 1 - \sum_{i \in P^+} (1 - \eta_i)$$

$$w \in \{0,1\}, \eta_i \in \{0,1\} \ i \in P^+.$$

As in the case of model $[M]$, the branching scheme implementation simplifies when $K^p = \lfloor \alpha \rfloor$ (resp. $L^p = \lceil \alpha \rceil$) takes his minimum (resp. maximum) value: If $K^p = 0$, one can set $\lambda_q = 0$ for all $q \in Q$ such that $a_q \geq p = a_{q^*}$ and the pricing subproblem modification reduces to adding to the subproblem formulation, the following set of constraints:

$$x_i \leq (p_i - 1) + (b_i - p_i + 1)\eta_i \quad i \in P^+$$

$$\sum_{i \in P^+} (1 - \eta_i) \geq 1$$

$$\eta_i \in \{0,1\} \ i \in P^+.$$

Note that, for all $i \in P^+$, $\sum_{q:a_q \geq p} \lambda_q \leq p_i \sum_{q:a_q \geq p} \lambda_q \leq \sum_{q:a_q \geq p} a_{i\,q} \lambda_q \leq \sum_q a_{i\,q} \lambda_q = b_i$. Now suppose that L^p takes its maximum value, i.e. $L^p = \min_{i \in P^+} \lfloor \frac{b_i}{p_i} \rfloor$. Moreover, if $\min_{i \in P^+} \lfloor \frac{b_i}{p_i} \rfloor = \min_{i \in P^+} \frac{b_i}{p_i}$, then $S^p = \{i \in P^+ : L^p = \frac{b_i}{p_i}\} \neq \emptyset$, and one can set $\lambda_q = 0$ for all $q \in \{q \in Q : \sum_{i \in S^p} a_{i\,q} > 0\} \backslash \{q \in Q : a_q \geq p\}$. Also, one can set $\lambda_q = 0$ for all $q \in \{q \in Q : a_q \geq p$ and $a_{i\,q} > p_i$ for some $i \in S^p\}$. The resulting subproblem modification reduces to adding to the subproblem formulation, the following set of constraints:

$$x_i = p_i \, w \quad i \in S^p$$

$$x_i \geq p_i \, w \quad i \in P^+ \setminus S^p$$

$$w \in \{0, 1\}.$$

3.5 Bounding

At every node of the branch-and-bound tree, a lower bound on the node master problem value Z^u can be obtained by solving its linear programming relaxation M^u_{LP} by means of column generation. In practice, however, solving the LP relaxation to optimality is cumbersome, as the column generation algorithm may require a large number of iterations to prove LP optimality. This drawback of the column generation technique is known as the *tailing-off effect*. Its consequences on the overall algorithm performance are far greater in an integer programming framework, since this can now potentially happen at each node of the branch-and-bound tree, and what is more the subproblem to be solved at each iteration is now typically a difficult integer program. In addition the subproblem becomes considerably harder to solve as the dual variables converge to their optimal values for the unrestricted master LP.

In an effort to control the tailing-off effect, we develop an alternate bounding strategy based on Lagrangean duality. Recall from Chapter 1, that the disaggregated formulation linear relaxation $[M_{LP}]$ is intimately linked to the Lagrangean dual. This fact has been traditionally exploited to derive lower bound at every iteration of the column generation algorithm (see for instance Vance et. al. (1992) [44]). Here, however, we improve upon the Lagrangean dual lower bound by dualizing only the set partitioning constraints(44) and the branching constraints defining $[M^u]$, and keeping the initial cardinality constraints (28-29) of model $[M]$.

Below, we formalize our claim in a proposition and then we show how the lower bounds can be exploited for early termination of the column generation algorithm. Finally, we push the argument one step further and show how to derive a priori bounds on the pricing subproblem value.

3.5.1 Valid node lower bound

At any given node u of the branch-and-bound tree, we apply column generation to solve the linear relaxation $[M^u_{LP}(Q)]$ of the node master problem (43). The intermediate values of the restricted master LP, $Z^u_{LP}(\tilde{Q})$, form a list of monotically non-increasing upper bounds on $Z^u_{LP}(Q)$. The following proposition shows that one can also compute a lower bound on $Z^u_{LP}(Q)$ at each iteration of the column generation procedure.

Proposition 5 *Let $(\pi, \mu, \nu) \in I\!R^m \times I\!R_-^{|G^u|} \times I\!R_+^{|H^u|}$ be an optimal dual solution of the restricted master LP, $M^u_{LP}(\tilde{Q})$, and let*

$$\begin{aligned} LB^u(\pi,\mu,\nu) \;=\; & Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0 \\ & + \min\{K^0(v^u(\pi,\mu,\nu)+\mu_0+\nu_0), L^0(v^u(\pi,\mu,\nu)+\mu_0+\nu_0)\} \end{aligned}$$

where $v^u(\pi, \mu, \nu)$ is the value of the pricing subproblem (47), then

$$LB^u(\pi,\mu,\nu) \le Z^u_{LP}(Q) \le Z^u_{LP}(\tilde{Q}) \tag{53}$$

Proof. Using (π, μ, ν) as weights, and dualizing all the constraints of $M^u_{LP}(Q)$ except for the initial cardinality constraints (28-29), gives

$$\begin{aligned} Z^u_{LP}(Q) \ge \quad & \textstyle\sum_{i\in I} \pi_i b_i + \sum_{g\in G^u_0} \mu_g K^g + \sum_{h\in H^u_0} \nu_h L^h \\ & + \min \textstyle\sum_{q\in Q} [c_q - \sum_{i\in I} \pi_i a_{i\,q} - \sum_{g\in G^u: a_q \ge a_g} \mu_g - \sum_{h\in H^u: a_q \ge a_h} \nu_h]\lambda_q \\ & s.t. \quad L^0 \le \textstyle\sum_{q\in Q} \lambda_q \le K^0 \\ = \quad & Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0 \\ & + \min \textstyle\sum_{q\in Q} [c_q - \sum_{i\in I} \pi_i a_{i\,q} - \sum_{g\in G^u: a_q \ge a_g} \mu_g - \sum_{h\in H^u: a_q \ge a_h} \nu_h]\lambda_q \\ & s.t. \quad L^0 \le \textstyle\sum_{q\in Q} \lambda_q \le K^0 \end{aligned}$$

where the last equality is obtained using (45). The remaining optimization problem immediately leads to the proposed lower bound. $Z^u_{LP}(Q) \le Z^u_{LP}(\tilde{Q})$ follows from

$\tilde{Q} \subseteq Q$. ∎

Every time the restricted master LP and the associated pricing subproblem are solved, one can compute the value of the lower bound $LB^u(\pi, \mu, \nu)$. Note that one can also use a lower bound on $v^u(\pi, \mu, \nu)$ (for instance the solution of a relaxation of the subproblem) to derive a valid lower bound on $Z^u_{LP}(Q)$. When the LP optimality condition holds, $v^u(\pi, \mu, \nu) = 0$, and $LB^u(\pi, \mu, \nu) = Z^u_{LP}(Q) = Z^u_{LP}(\tilde{Q})$. In Figure 9, we sketch the evolution of the values of $LB^u(\pi, \mu, \nu)$ and $Z^u_{LP}(\tilde{Q})$. During the course of the column generation algorithm, $Z^u_{LP}(\tilde{Q})$ monotonically decreases towards $Z^u_{LP}(Q)$, while $LB^u(\pi, \mu, \nu)$ increases but not monotonically towards $Z^u_{LP}(Q)$. The flat end of the curves represent the tailing off effect which we typically observe. The

Figure 9: Bounds evolution during the column generation procedure.

best previously generated bound (i.e. the largest value of $LB^u(\pi, \mu, \nu)$) is kept as node lower bound LB^u. We initialize LB^u with the lower bound that was computed at the predecessor node, and we update it at every column generation iteration.

The interest of the node lower bound LB^u is twofold. First, if column generation is interrupted before LP optimality is proved, one still has a lower bound on the node problem value. Second, the lower bound leads to criteria for early termination of the column generation algorithm.

3.5.2 Early termination of column generation at a node

At any given node u, column generation is stopped when $Z^u_{LP}(Q)$, the value of the master LP, is known. Up to now, we have only used the fact that $Z^u_{LP}(Q) = Z^u_{LP}(\tilde{Q})$ when the LP optimality condition (i) holds,

$$(i) \quad v^u(\pi, \mu, \nu) \geq 0 \tag{54}$$

Proposition 5 provides another way to prove that $Z^u_{LP}(Q)$ is known. That is checking if the gap between the lower and upper bounds on $Z^u_{LP}(Q)$ is closed, i.e.,

$$(ii) \quad LB^u = Z^u_{LP}(\tilde{Q}) \tag{55}$$

where LB^u has been defined previously as the maximum of the predecessor node lower bound and all previously computed lower bounds $LB^u(\pi, \mu, \nu)$ at node u. Note that stopping criteria (i) and (ii) are equivalent when $LB^u = LB^u(\pi, \mu, \nu)$, which only happens when the last computed lower bound is the best one. Otherwise, due to degeneracy of the master LP, we may have that (i) does not hold but (ii) does hold.

Moreover, we can also stop column generation when the current lower approximation LB^u of $Z^u_{LP}(Q)$ is high enough to prune the node. Letting Z^{INC} be the current incumbent primal integer solution value,

$$(iii) \quad LB^u \geq Z^{INC} \tag{56}$$

implies that, at node u, there is no integer solution better than the current incumbent solution and the node can be pruned.

If we assume that the optimal solution value Z of $[M]$ is integral, as in many application, the above stopping criteria become even tighter. Under the integrality assumption, the lower bound produced at a given node u on the master problem value Z^u is $\lceil Z^u_{LP}(Q) \rceil$. Thus we stop column generation when either we know the value of $\lceil Z^u_{LP}(Q) \rceil$, or its current lower approximation $\lceil LB^u \rceil$ is high enough to prune the node. This amounts to checking 3 stopping conditions:

$$\begin{array}{lrcl} (i) & v^u(\pi, \mu, \nu) & \geq & 0 \\ (ii) & \lceil LB^u \rceil & \geq & Z^u_{LP}(\tilde{Q}) \\ (iii) & \lceil LB^u \rceil & \geq & Z^{INC} \end{array}$$

If case (i) holds, $Z^u_{LP}(Q) = Z^u_{LP}(\tilde{Q})$, and thus the node bound is $\lceil Z^u_{LP}(\tilde{Q}) \rceil$. Case (ii) implies $\lceil Z^u_{LP}(Q) \rceil = \lceil LB^u \rceil$, since $\lceil LB^u \rceil \leq \lceil Z^u_{LP}(Q) \rceil \leq \lceil Z^u_{LP}(\tilde{Q}) \rceil \leq \lceil LB^u \rceil$. Case (iii) implies $Z^u \geq Z^{INC}$, since $Z^{INC} \leq \lceil LB^u \rceil \leq \lceil Z^u_{LP}(Q) \rceil \leq \lceil Z^u \rceil = Z^u$.

3.5.3 A priori bound on the pricing subproblem value

The column generation stopping conditions presented in the previous section can be used to derive an a priori upper bound on the subproblem value. We shall show that if, at a given column generation iteration, the minimum reduced cost is not smaller than the a priori bound, then one of the stopping conditions holds and either the node bound is known or the node can be pruned.

Stopping condition (i) induces a trivial bound on the reduced cost of a newly generated pattern. If, in conditions (ii) and (iii), we replace LB^u by $LB^u(\pi, \mu, \nu)$, we obtain valid stopping conditions whose expressions explicitly contain $v^u(\pi, \mu, \nu)$. A simple reformulation of these expressions leads to bounds on $v^u(\pi, \mu, \nu)$. Next we perform these developments under the integrality assumption. In the general case, similar developments lead to an a priori bound on the pricing subproblem as well.

Let $\rho_1 = \max\{\frac{\alpha_1}{K^0}, \frac{\alpha_1}{L^0}\} - \mu_0 - \nu_0$ where

$$\alpha_1 = \lceil Z^u_{LP}(\tilde{Q}) \rceil - 1 - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$$

and $\rho_2 = \max\{\frac{\alpha_2}{K^0}, \frac{\alpha_2}{L^0}\} - \mu_0 - \nu_0$, where

$$\alpha_2 = Z^{INC} - 1 - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$$

and $\rho = \min\{\rho_1, \rho_2\}$.

Proposition 6 *If $v^u(\pi, \mu, \nu) > \rho$, column generation terminates. If in addition $\rho = \rho_2$, node u can be pruned.*

Proof. Condition (ii) implies that the lower bound for node u is known if $\lceil LB^u(\pi, \mu, \nu) \rceil \geq Z^u_{LP}(\tilde{Q})$. This is equivalent to the condition $LB^u(\pi, \mu, \nu) > \lceil Z^u_{LP}(\tilde{Q}) \rceil - 1$ or written explicitly:

$$Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0 + \min\{K^0(v^u(\pi, \mu, \nu) + \mu_0 + \nu_0), L^0(v^u(\pi, \mu, \nu) + \mu_0 + \nu_0)\}$$

$$> \lceil Z^u_{LP}(\tilde{Q}) \rceil - 1$$

or

$$K^0(v^u(\pi,\mu,\nu)+\mu_0+\nu_0) > \lceil Z^u_{LP}(\tilde{Q})\rceil - 1 - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$$

and

$$L^0(v^u(\pi,\mu,\nu)+\mu_0+\nu_0) > \lceil z^u_{LP}(\tilde{Q})\rceil - 1 - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$$

This in turn can be written as:

$$v^u(\pi,\mu,\nu) > \frac{\alpha_1}{K^0} - \mu_0 - \nu_0$$

and

$$v^u(\pi,\mu,\nu) > \frac{\alpha_1}{L^0} - \mu_0 - \nu_0.$$

Thus $v^u(\pi,\mu,\nu) > \rho_1$ implies condition (ii) holds. The other case is identical using stopping condition (iii) in place of (ii). If $v^u(\pi,\mu,\nu) > \rho_2$, condition (iii) holds and the node can be pruned. ∎

In conclusion, each stopping condition leads to a bound on $v^u(\pi,\mu,\nu)$. The minimum one is kept as an a priori upper bound on the subproblem. If this bound is violated by the minimum column reduced cost, column generation terminates. The interest of such a bound is to potentially speed up the solution of the subproblem. Using the a priori bound, one can define an initial upper cutoff value for the subproblem (or add an explicit constraint in the formulation of the subproblem); if the subproblem becomes infeasible, a stopping condition holds. We observed in our computations that, as we approach optimality of the master LP at a node, this a priori bound becomes tighter and thus may help in reducing the tailing-off effect.

4 Implementation of IP Column Generation

Implementing an algorithm that combines column generation with branch-and-bound to solve model $[M]$ raises many issues. One faces questions like: how to initialize the column generation algorithm, how to solve the pricing subproblem (exact versus approximation algorithm), what kind of column generation strategy to adopt (multiple versus single column generation at each iteration), how to implement branching efficiently, etc... From the judicious choice of implementation will depend the efficiency of the algorithm. In our experience, the computational time to solve a particular instance of the network design application decreased from 529 to 107 seconds of CPU time between the initial straightforward implementation of IP column generation presented in Chapter 3 and the enhanced later version of the same algorithm.

In this Chapter, we discuss the issues concerning the implementation of the IP column generation algorithm. We compare different possible implementation strategies. We describe our own implementation. We share the insights that we have acquired through computational experiments. We use partial computational results to illustrate the impact of some implementation choices. We also refer to computational experience reported in the literature. After discussing our implementation strategies, we present a flow sheet of the overall algorithm.

4.1 Partitioning versus Covering Formulation

Although we have presented the IP column generation algorithm for the set partitioning formulation $[M]$, everything carries over to set covering and set packing formulations. The sign of the dual variables associated with the linking constraints is then restricted.

The set covering (resp. packing) formulation admits more solutions. However, for the applications we consider, the objective value of the optimal solution is identical to that of the set partitioning optimal solution. And, if in an optimal set covering (resp. packing) solution, an element i is covered more (resp. less) than b_i times, it can be removed from one of the feasible patterns selected in the solution (resp. added as a feasible pattern on its own) at no extra cost (resp. profit) in a post-processing phase. For instance, for the network design application, the real problem is to assign each demand (edge) to the network at least once. If, in the proposed

solution, an edge is assigned to more than one ring, it can be removed from one of the rings according to the designer's choice.

In general, as long as any sub-pattern of a feasible pattern is another feasible pattern without a higher cost, adopting the set covering formulation instead of the partitioning formulation is a matter of implementation choice. The branching scheme that we have designed for the set partitioning formulation might eliminate some feasible integer set covering (resp. packing) solutions. But this has no effect on the search for an optimal solution since there exists a set covering (resp. packing) optimal solution among the set partitioning solutions.

In our implementation, we opt for the set covering (resp. packing) formulation (see Appendix A) for the following reasons:

- The main reason is that it is easier to find an initial master LP feasible solution, as we shall see.
- The dual variables are then restricted in sign, which always leads to the same cost structure in the pricing subproblem and thus makes it easier to develop a heuristic solution method for the subproblem.
- In the literature, the set covering linear program is reported to be numerically more stable and easier to solve than the set partitioning LP (see Barnhart et. al. (1994) [4]).
- We avoid column generation iterations whose only outcome is to generate a pattern that is a subset of an already existing pattern (resp. to generate a zero profit pattern).
- It increases our chances to find master integer solutions combining columns from the restricted formulation.

The two last points merits more explanation. With the partitioning formulation, the dual variables associated with a row may be negative, for instance suppose $\pi_i < 0$. Assume a column q currently in the master with a non-negative reduced cost $0 \leq \overline{c}_q < -\pi_i$ has an entry 1 in row i ($a_{i\,q} = 1$). Then, by removing element i from that column, we create a new column q' that prices out negatively even if $c_q = c_{q'}$. For instance, using a set partitioning formulation for the network design application, we observed the following: when the current formulation contained a pattern that includes a clique of 3 edges at cost 3, one of which has now a negative

dual price, a new column was generated by removing that edge; the cost of the new pattern was identical to the old because one still needs to install 3 multiplexers.

There is more chance of having intermediate integer solutions during the solution of the LP master by column generation with the set covering (resp. packing) formulation. Indeed, the set covering (resp. packing) constraints are not as restrictive as the set partitioning constraints, so one can construct more feasible integer solutions from the existing columns. Solving the restricted master IP by branch-and-bound will also provide better heuristic solutions if one uses the set covering (resp. packing) formulation.

To enlarge the set of feasible integer solutions of the covering formulation, we try to generate maximal patterns, that is, after a new pattern is generated, we add elements that can be included in this new pattern without augmenting the cost, barring capacity. For the set packing formulation, one eliminates an element from a pattern if this element makes a zero contribution to the profit of that pattern.

4.2 Initial Set of Columns (Phase 1)

The column generation procedure must be initialized with a feasible solution of the master LP (or infeasibility of the master problem must be proved). This issue is addressed by providing an initial set of columns for the restricted master formulation at each node of the master branch-and-bound tree. In this Section, we discuss the choice of initial columns. The primary purpose of this initialization is to provide a feasible master LP solution. However, the initial set of of columns also determines the starting value of $Z^u_{LP}(\tilde{Q})$, an initial incumbent solution if this set of columns defines an integer solution, and initial bounds on the dual prices (cfr dual formulation of the master problem presented in Appendix A). Thus, with an appropriate initial set of columns, one can get a good start in the column generation procedure.

Generating an initial feasible solution for the master LP might require the solution of a phase 1 problem, i.e., to initialize the formulation with appropriate artificial variables and an artificial objective function that penalize the use of artificial variables in the solution. However, if we use an artificial objective function, we may be generating columns that will serve only for the phase 1 because their cost is too high to be part of the solution during phase 2.

Instead, we combine phase 1 with phase 2, that is we add artificial variables in the formulation and we keep the original master objective function, to which we add costs associated with the selection of artificial variables. Moreover, we try to keep the costs of the artificial variables as small as possible so that the shadow prices are not too much the result of the artificial variable costs but they are driven by the costs of the real columns. If column generation terminates with a solution that includes some artificial variables, then we augment their cost and return to column generation. Doing so, we either end up with a solution free of artificial variables or the node problem lower bound increases to the point at which the node gets pruned, assuming we initialized the algorithm with a cut-off value (i.e. a bound on the cost of a master integer solution).

At the root node, the current restricted master LP may be infeasible because the original problem is infeasible or simply because the relevant columns have not yet been included in the formulation. At successor nodes, infeasibilities of the restricted master LP can be caused by the branching constraints or be due to the absence of relevant columns. Recall that in the Ryan and Foster branching scheme (which is the one we mainly use), the branching constraints are explicitly enforced by removing columns from the formulation. So, infeasibilities due to branching can appear as infeasibilities due to the absence of columns necessary to form a feasible solution.

In any case, the restricted master LP augmented with artificial variables will be feasible. If the unrestricted master is infeasible, the artificial variables will always remain in the solution of the augmented restricted master LP, so our scheme will lead to pruning the node by bound.

In practice, one needs only one artificial variable to make the covering (resp. packing) formulation of the restricted master LP $M^u_{LP}(\tilde{Q})$ feasible. This artificial pattern has an entry equal to one in all rows corresponding to greater or equal constraints, and zero for the rows corresponding to less or equal constraints. Its cost may be set equal to an a priori upper bound on Z. Note that it is not so easy to initialize the set partitioning formulation.

To detect obvious infeasibilities due to the branching constraints, we check if these are consistent. For instance, a set of branching constraints saying that elements i and j must be assigned to the same pattern, elements j and k must also be assigned to a same pattern, but elements i and k must be assigned to different patterns, is

inconsistent. Or if i and j are forced to be assigned to the same pattern but their total weight exceeds the maximal feasible weight in a pattern, branching is inconsistent. These tests are carried out while forming the group of elements that are forced to be in the same pattern due to the branching constraints. If infeasibility is detected in this way, the node is pruned.

At the root node, the choice of initial columns for the master formulation is application specific. In addition to the artificial variable, we include a set of columns that define initial bounds on the dual variables. For the network design applications (ND and NDSA, cfr Appendix A.1 and A.2), we initially include in the master the unit matrix, that is $|E|$ patterns containing a single edge. This provides an initial feasible solution barring the cardinality constraints, and bounds the dual variables: $\pi_i + \pi_j \leq c_e$ for all $e = (i, j)$.

For the clustering application (CLUST, cfr Appendix A.3), we initialize the formulation with columns containing a single edge (i.e. patterns q with $a_{i\,q} = a_{j\,q} = 1$ for $e = (i, j) \in E$ such that $d_i + d_j \leq C$) so that the initial shadow prices are not trivially zero: $\pi_i + \pi_j \geq c_e$. Feasibility is not really a problem with the set packing formulation. In our experiments, we have never needed to use an artificial variable.

For the single-machine multi-item lot-sizing application (MILS, cfr Appendix A.4), we initialize the formulation with one column for each item that corresponds to producing this item in each period for which there is a positive demand. Assuming that $d_t^i \leq U^t$ for all t, as is the case for our data set, the i^{th} pattern is defined by $a_{t\,i} = 1$ if $d_t^i > 0$ for all $t = 1, \ldots, T$ and $a_{(T+i)\,i} = 1$. This set of columns orients the initial shadow prices. Here, we need the artificial variable as well. An alternative choice of artificial variables consists of having an artificial pattern for each item with a single entry of one in the row corresponding to the covering of that item (i.e. $a_{t\,q} = 0\ \forall t$ and $a_{(T+i)\,q} = 1$).

At successor nodes, for all applications, we initialize the node master problem restricted formulation using <u>all</u> previously generated columns that satisfy the branching constraints. Also, we keep the artificial variables in the formulation at all times and reset their cost to their initial values every time we start processing a new node. Note that, since we do not remove an artificial variable that does not satisfy the branching constraints from the formulation, it is important to complete the phase 1 (i.e. iterate on the column generation procedure until no artificial variables are

part of the solution) before branching. For otherwise, we might branch using the same rule as at a predecessor node and the branch-and-bound algorithm might not terminate.

4.3 Branching Selection

Our branching scheme for problem $[M]$ is based on Proposition 3 which guarantees that we can eliminate all fractional solutions by choosing a subset of elements $T \subseteq I$ that leads to a fractional subset sum (36) and imposing branching constraints like (38) and (39). We describe here how, in practice, we select a subset T on which to branch.

For the network design application (ND), we consider three types of branching (i.e. of subset T): cardinality branching, node cardinality branching, and branching on pairs of edges. Cardinality branching consists of taking T to be the empty set; it amounts to checking if the total number of patterns used in the solution (i.e. $\sum_{q \in Q} \lambda_q$) is integer. The resulting branching constraints aim at forcing the master solution to use an integer number of patterns. To implement it, one simply needs to modify K^0 on one branch and L^0 on the other, while the subproblem remains unchanged. These global cardinality constraints enforced in the master augment our chances to have intermediate integer solutions during column generation. Tighter values for K^0 and L^0 improve the quality of the bound $LB^u(\pi, \mu, \nu)$ whose value depends on K^0 and L^0, and lead to better a priori bounds on the subproblem value.

If the master solution is fractional but $\sum_{q \in \tilde{Q}} \lambda_q$ is integer, we then search for a subset T of two edges e and f such that $\sum_{q \in \tilde{Q}: a_{e\,q} = a_{f\,q} = 1} \lambda_q$ is fractional. For every node i, we try all possible pairs of edges from the set of edges incident to node i, $\delta(i) \subseteq E$. We keep the pair that leads to the most fractional subset sum (36) (i.e. whose fractional part is the closest to $\frac{1}{2}$). We favor edges with high demand. The reason for this selection of edge pairs is heuristic: we try to derive branching constraints that help in constructing good integer solutions. Then, in one (resp. in the other) successor node, we remove all columns such that $a_{e\,q} = a_{f\,q} = 1$ (resp. $a_{e\,q} \neq a_{f\,q}$) and we modify the subproblem accordingly by explicitly adding to its formulation the extra constraint $x_e + x_f \leq 1$ (resp. $x_e = x_f$).

In our experimentation with the network design problem ND, we have been able to eliminate all fractional solutions using only such subsets of pairs of edges. However,

we have observed that the branch-and-bound tree grows bigger as the average edge demand size gets smaller relative to the ring capacity. Indeed, as the granularity of demand increases, the number of master feasible solutions gets larger and forcing two edges together or apart might not be very restrictive. In this case, we have obtained smaller branch-and-bound trees by using a third kind of branching that imposes more aggregated restrictions.

The third type of branching is obtained by choosing $T = \{i\}$ where $i \in V$ is a node. This selection of T amounts to checking that the number of selected patterns for which a multiplexer is installed at node i is integer. If we define $r_{iq} = 1$ if pattern q contains node i and zero otherwise (i.e. $r_{iq} = 1$ if $\sum_{e \in \delta(i)} a_{eq} > 0$), then $\sum_{q \in \tilde{Q}: r_{iq}=1} \lambda_q = \alpha$ is the number of multiplexers installed at node i. When this number α is fractional, one can branch by adding explicitly to the master

$$\sum_{q \in \tilde{Q}: r_{iq}=1} \lambda_q \leq \lfloor \alpha \rfloor \tag{57}$$

on one branch and

$$\sum_{q \in \tilde{Q}: r_{iq}=1} \lambda_q \geq \lceil \alpha \rceil \tag{58}$$

on the other branch. In the subproblem, one needs only to modify the cost of installing a multiplexer at node i (which is originally 1) in order to account for the new dual variables. We call this branching rule node cardinality branching.

Branching on the number of patterns that include a given node leads to more aggregated (global) restriction than branching on a pair of edges. When we apply this type of branching, we observe a smaller branch-and-bound tree for problems with demands small relative to the capacity. However, node cardinality branching makes the master and the subproblem harder to solve. So the overall performance (CPU time) of the algorithm, using node cardinality branching, improves only for instances exhibiting high demand granularity.

They are other possible types of branching which we have not implemented to date. For instance, one can select $T = \{e, i\}$ with $e \in E$, $i \in V$, and $e \notin \delta(i)$. Forcing e and i to be on the same ring corresponds to trying to assign edge e with some edges incident to i. Imposing that e and i must be on different rings means that e and f must be apart for all $f \in \delta(i)$.

One may also use sets $T = \{i, j\}$ where $i, j \in V$ are nodes not linked by an edge. So there is an whole range of possible branching rules, from the more global to the

more specific: cardinality, node cardinality, pairs of nodes, pairs consisting of one node and one edge, pairs of edges. The latter subclass (pairs of edges) corresponds to the straightforward implementation of Ryan and Foster branching scheme. It is thus sufficient in theory to eliminate all fractional solutions.

For the network design application with split demand assignment (NDSA), the right hand side of the master covering constraints can take value 2. Proposition 3 tells us that we may have to try subsets T of cardinality 3 in order to identify a subset sum (36) on which to branch. However, as long as there is a fractional λ_q corresponding to a pattern q for which $a_{e\,q} = 1$ for some e with $b_e = 1$, then the Ryan and Foster Branching scheme can be used. In our computational experiments, we have always been able to eliminate all fractional master solution using cardinality branching, node cardinality branching, or branching on pairs of edges one of which had a corresponding right hand side $b_e = 1$.

For the clustering problem (CLUST), we branch using cardinality branching first (optional) and eliminating the remaining fractional solutions by branching on edges, i.e., on pairs of nodes. We search for a set $T = \{i,\, j\}$ with $e = (i, j) \in E$ that leads to the most fractional subset sum (36). We break ties by selecting the edge with maximum cost.

For the multi-item single-machine lot-sizing problem (MILS), we branch on sets $T = \{i,\, t\}$ involving an item i and a period t for which $\sum_{q \in \tilde{Q}: a_{(T+i)\,q} = a_{t\,q} = 1} \lambda_q$ is the closest to $\frac{1}{2}$. As mentioned in the last Chapter, these subset sums correspond to the setup variables in the original formulation. It is thus sufficient to branch on these subsets T in order to eliminate all fractional solutions.

4.4 Solving the Pricing Subproblem

Through the decomposition approach, the solution of the original integer program boils down to solving many pricing subproblems. In integer programming column generation, the pricing subproblem is often itself a difficult integer program. If the decomposition approach performs well, it is because the difficulty of solving the original problem has been replaced by tackling a somewhat simpler but still difficult subproblem. In comparison, when the pricing subproblem LP relaxation has the integrality property making the subproblem optimization easy, the master LP bound is no tighter than the one obtained by solving the linear relaxation of the original formulation $[F]$. In the applications we consider, most of the work of the

algorithm is spent in solving the subproblems. This computationally intensive part of the algorithm requires over 90 % of the CPU time.

Note that the column generation algorithm does not require an optimal solution of the pricing subproblem. In a minimization (resp. maximization) problem, any column with negative (resp. positive) reduced cost can be used to move on to the next iteration of the column generation procedure. Moreover, as attested by recent studies on the simplex pivot selection criteria (see for instance Forrest and Goldfarb (1992) [19]), using a maximum absolute value reduced cost column as entering variable does not necessarily lead to the fastest converging linear programming solution method for the master. The selection of entering columns leading to good convergence performance is discussed in the next section. Let us now focus on how we generate new patterns.

The solution of the application specific pricing subproblem is a research topic on its own. For many applications however, some results concerning the subproblem can be found in the literature. To efficiently tackle this bottleneck of our algorithm, we combine the use of bounds, an approximation algorithm, and strengthening of the formulation.

Considering the minimization pricing subproblem (47) defined for model $[M]$, one can derive a lower bound on its value by solving its linear programming relaxation. We try to improve this lower bound by strengthening the formulation. That is, we use a cutting plane algorithm. If we find an integer solution to the LP relaxation of the pricing subproblem, it is optimal. On the other hand, we compute an upper bound on the minimum reduced cost by solving the pricing subproblem heuristically. If the heuristic cost is equal to the pricing subproblem lower bound, the heuristic solution is optimal. Otherwise, if we need to solve the pricing subproblem optimally, we apply standard branch-and-bound to the strengthened formulation, using some priority rule for branching.

To test for early termination of the column generation algorithm, we use the a priori upper bounds on the subproblem value developed in Section 3.5.3. The specific expressions for these bounds are given for each application in Appendix A. We define an *admissible* new pattern as a feasible solution of the pricing subproblem whose reduced cost satisfies the a priori bounds.

At every iteration of the column generation procedure, we first compute the a priori upper bounds on the pricing subproblem. Then, we solve the pricing subproblem approximately, using an application specific heuristic. We may find a few patterns whose reduced cost satisfies the a priori bounds (i.e. admissible patterns). If not, we approximate the subproblem value from below: we compute progressively tighter lower bounds starting with the linear programming relaxation, then moving to the cutting plane algorithm and, as a last resort, to the branch-and-bound algorithm. This three phase lower bound computation is interrupted by optimality, if we obtain an integer solution, or by bound, if the current lower bound violates one of the a priori subproblem upper bounds.

At the end of this search for new patterns, we either have proved that column generation may be terminated; or we have generated one or more admissible patterns (i.e. patterns of appropriate reduced cost). We then add some of these new admissible patterns to the master formulation depending on our column selection criteria which we discuss in the next section.

Let us now turn to the application specific aspects of this process, namely the heuristic for the subproblem and the cutting plane algorithm. For the network design applications, we have developed a greedy heuristic for the subproblem. We first define groups of edges that must be assigned together according to the branching constraints. Then, we construct a feasible assignment of groups of edges to a ring: we initially assign a specific group to the ring; then we sequentially try to add a new node on the ring; we see what groups of edges can then be accommodated on the ring (i.e. a group whose edges are incident to nodes already on the ring, that is not excluded because of edge disjunctive branching constraints, and whose capacity requirement does not exceed the remaining ring capacity); we compute the resulting cost; we record the node selection that leads to the most negative cost difference; and we implement this node addition. Each pattern initialization by a group of edges may lead to a different negative reduced cost column. We record all the new admissible columns.

For the clustering application, we use a similar subproblem heuristic. We construct a feasible cluster by initially selecting a group of nodes as defined by the branching constraints; we then iteratively add the group of nodes that leads to the best profit improvement. A group of nodes can be added to a cluster if its weight does not exceed the remaining cluster capacity and if it is not excluded by another group

already in the cluster. Note that, since clusters are complete subgraphs, the node selection determines the edges that are within the cluster. For the multi-item single-machine lot-sizing application, we did not use any heuristic for the subproblem.

To develop a full-blown cutting plane algorithm for the pricing subproblem of each application would have required specific polyhedral studies. Instead, we use existing results. For the network design subproblem, we use knapsack cover inequalities. For the clustering subproblem, we based our cutting planes on cuts developed by Johnson et al. ([29]) and de Souza ([13]). In both these applications, however, we obtain a faster algorithm by using standard branch-and-bound on the initial subproblem formulation. This may be due to the fact that the cuts we add are not strong enough or/and because, for this size of problems, standard branch-and-bound is more efficient.

For the single-machine multi-item lot-sizing application, we solve the pricing subproblem optimally at every iteration of the column generation algorithm. We rely on the valid inequalities developed by Constantino [7] to strengthen the formulation. Indeed, standard branch-and-bound applied to the initial formulation performs poorly.

An interesting issue raised by the use of a cutting plane algorithm for the subproblem is the management of cuts. In the course of the algorithm, we solve many subproblems which differ only in their objective coefficients and in some extra constraints due to branching. So the globally valid cuts, generated while solving a subproblem, may be recycled for other subproblem solutions. On the other hand, if we leave all the cuts in the subproblem formulation all the time, the size of the formulation quickly increases, affecting the solution time.

In our implementation, we clean up the formulation (i.e. remove all the cuts), either after each subproblem solution, or, after each master node problem termination. In addition, we keep all the generated cuts in a pool (one for each item in the case of the MILS application) which we scan before calling the cut generation subroutine. These enhancements lead to a marginal reduction in computation time.

4.5 Column Selection Strategy

The implicit question when one establishes a strategy for selecting columns is: what is a *good* column. In other words, what columns serve our goals better. Recall that

the objective of the algorithm is twofold: on the one hand, to produce an integer primal master solution, and on the other hand, to provide a bound attesting to the quality of this solution. So a column can be qualified as good because it helps in constructing an integer solution or because it helps in solving the master linear relaxation faster. A column selection strategy can aim at generating good columns and/or at selecting the good ones from among all the columns that have been generated at a given iteration.

Good columns for the LP optimization:

In an attempt to understand what columns to add in the master formulation in order to obtain fast convergence of the LP optimization, one can consult two bodies of literature. First, the research on pivot selection for the simplex method (see for instance Forrest and Goldfarb (1992) [19]) studies the issue of optimum reduced cost versus steepest edge criteria for selecting the entering column. Second, the research on developing an efficient cutting plane algorithm (Balas et. al. (1993) [2]) studies cut selection criteria based on greatest violation versus deepest cut. Both domains are linked since adding columns in the primal master corresponds to adding cuts in the dual formulation of the master. Let us examine the pros and the cons of these column selection criteria, starting with the optimal reduced cost criteria.

At a given iteration of the column generation algorithm for solving the linear master problem $[M_{LP}]$, the most negative reduced cost column represents the most violated cut in the dual formulation. The logic for selecting a most negative reduced cost column is that it might lead to the best objective improvement. However, there is no guarantee that this will effectively happen. For instance, it could be that the current objective value is optimal and that, due to degeneracy, some columns price out negatively although no objective improvement is possible. So, is there some other column selection criteria that would not have this drawback?

The deepest cut criteria is based on a geometric argument. It does not measure the progress made because of the entering column in terms of objective improvement but in terms of restriction of the dual polyhedron. A good column is one that cuts a big chunk out off the current approximation of the dual master polyhedron. The deepest cut (the one that maximizes the Euclidian distance to the current dual solution) potentially leads to the best restriction of the dual feasible space.

The Euclidian distance between the cut and the current dual solution $(\hat{\pi}, \hat{\mu}, \hat{\nu}) \in$

$\mathbb{R}^n_+ \times \mathbb{R}^1_- \times \mathbb{R}^1_+$ is the distance between this latter point and its projection on the hyperplane defined by the cut. Let (c_q, a_q) be a column that prices out negatively for the current value of the dual variables, i.e. a column that cuts off the current dual solution:

$$\hat{\pi}\, a_q \;+\; \hat{\mu} \;+\; \hat{\nu} \;>\; c_q \;. \tag{59}$$

Let $(\tilde{\pi}, \tilde{\mu}, \tilde{\nu})$ be the projection of $(\hat{\pi}, \hat{\mu}, \hat{\nu})$ on the hyperplane defined by $\pi\, a_q \;+\; \mu \;+\; \nu \;=\; c_q$. That is

$$\tilde{\pi}\, a_q \;+\; \tilde{\mu} \;+\; \tilde{\nu} \;=\; c_q \;. \tag{60}$$

and

$$(\tilde{\pi}, \tilde{\mu}, \tilde{\nu}) \;=\; (\hat{\pi}, \hat{\mu}, \hat{\nu}) \;+\; \alpha\,(a_q, 1, 1) \tag{61}$$

for some scalar α. The Euclidian distance from $(\hat{\pi}, \hat{\mu}, \hat{\nu})$ to the cut is:

$$||(\tilde{\pi}, \tilde{\mu}, \tilde{\nu}) - (\hat{\pi}, \hat{\mu}, \hat{\nu})|| \;=\; \frac{\hat{\pi}\, a_q \;+\; \hat{\mu} \;+\; \hat{\nu} \;-\; c_q}{||(a_q, 1, 1)||} \tag{62}$$

that is the violation divided by the norm of the vector defining the cut. Note that this dual measure of the deepest cut is related to the primal measure of the steepest edge (see Forrest and Goldfarb (1992) [19]).

This geometric approach for evaluating the quality of a column also has its drawbacks. First, generating a column that maximizes the distance between the current dual solution and its projection is computationally expensive. In our case, since $a_q \in \{0,1\}^n$, a brute force implementation consists of solving a subproblem for each possible value of the norm of $(a_q, 1, 1)$. Second, this measure become totally irrelevant if the projected point, $(\tilde{\pi}, \tilde{\mu}, \tilde{\nu})$, is not dual feasible. For instance, if $(\tilde{\pi}, \tilde{\mu}, \tilde{\nu})$ does not belong to $\mathbb{R}^n_+ \times \mathbb{R}^1_- \times \mathbb{R}^1_+$, the above criteria might lead to the selection of a column that cuts off much less of the dual space than other columns with smaller Euclidian distance to the current dual solution. The third drawback of this column selection criteria also applies to the most negative reduced cost criteria: in fact, these measurements are very sensitive to the current value of the dual solution. But, there are typically many alternative optimal dual solutions to the restricted master. Because of all these drawbacks, we did not implement a column selection rule based on the deepest cut criteria.

Selecting a dual solution:

From the previous discussion we remember that the analysis of the effect of an additional column on the dual space geometry is a valuable tool to characterize the quality of this column; and that column reduced costs depend on the current set

of dual values. Typically, the restricted master LP admits many alternative dual solutions all of which lie on a face of the dual polyhedron. Instead of considering an extreme point of this face, we could take the central point as dual optimal solution. Then, a column that cuts off such a dual solution is more likely to cut a bigger portion of the dual solution space.

We have carried out limited computational experiments along these lines. For the clustering application (cfr Appendix A.3), we try to generate *pseudo*-central points as dual solutions of the restricted master LP at each iteration of the column generation algorithm. That is, we solve the master LP twice: once to get the objective value; and a second time to find, from among all dual solutions leading to that objective value, a dual solution that maximizes the sum of auxiliary variables v_q which represent the minimum value of a variable π on the face defined by column q (i.e. $v_q \leq \pi_i$ for all i such that $a_{i\,q} = 1$). We have observed that as a consequence of this choice of dual variables, the pricing subproblem became much harder to solve (the heuristic for the subproblem very often fails to produce a positive reduced cost column, and we then have to use branch-and-bound to solve the subproblem). So the overall CPU time of the algorithm increases by 56% on average.

We also see good reasons for using extreme point dual solutions. With very different dual solutions, one generates different types of columns which, hopefully, are complementary for the construction of good primal solutions and whose convex combination may encompass a larger master LP feasible space. Moreover, extreme points being easier to cut, the column generation subproblem is solved faster. When the column generation procedure is tailing off, solving the subproblem optimally at each iteration to try to prove LP optimality, one could use central point dual solution and hopefully reduce the number of iterations. However, we have not implemented such a two-phase procedure for selecting dual solutions. In practice, we use the extreme point dual solutions provided by the LP solver.

Dominated/undominated columns:

Another column generation selection criterion that we use to improve the master LP convergence is based on a simple domination concept. We say that a column k of model $[M]$ is *dominated* if there exists another column $l \in Q$, such that the reduced cost of k, $\overline{c_k}$, is greater or equal to that of l, $\overline{c_l}$, for all dual values $(\pi, \mu, \nu) \in I\!R^n_+ \times I\!R^1_- \times I\!R^1_+$. On the other hand, a column k is *undominated* if for all columns $q \in Q$, there exists a set of dual variables $(\pi, \mu, \nu) \in I\!R^n_+ \times I\!R^1_- \times I\!R^1_+$

for which $\overline{c_k} < \overline{c_q}$. For instance, considering the network design application (ND), a pattern to which an edge can be added without having to install another multiplexer is dominated. If the edges in a pattern form a clique, the pattern is undominated.

If we discover that a column is dominated, we replace it by the column that dominates it. This is done in the post-processing phase described in Section 4.1 by augmenting (resp. reducing) a column of the covering (resp. packing) formulation whenever it is possible. Also, in the network design and clustering applications, the columns we use to initialize the master formulation and to define initial bounds on the dual variables (see Section 4.2) are undominated columns (cliques).

Good columns for IP optimization:

In an effort to select columns that improve our chances to find good integer master solutions, we have experimented with different strategies. In previous research (Vanderbeck (1993) [45]), we have tried to initialize the master formulation with an optimal integer solution and noted that the algorithm does not perform better. For the network design application, we have tried to initialize the master formulation with a set of columns that are good candidates to be part of an optimal integer solution (for instance, patterns corresponding to a selection of edges that use almost all the ring capacity and exhibit high node commonality). The performance of the resulting algorithm is worse.

Our understanding is that columns which are good for integer solutions are not helpful for the master LP solution (in the next chapter, we present computational results supporting this claim). As far as LP optimization is concerned, the column generation algorithm itself seems to be the best mechanism for generating appropriate columns. By generating extra columns which are supposedly good for the IP optimization, we perturb this mechanism (influencing the values of the dual variables), and we unnecessarily increase the size of the formulation, slowing down the master LP solution. Moreover, for our applications, the performance of the overall algorithm depends mainly on the solution efficiency of the LP master (remember that solving the master LP requires solving many difficult column generation subproblems). The integer solution only helps by providing a cut off value to prune the branch-and-bound tree. So the column selection aims first at helping the master LP solution, and, as a secondary criteria (to break ties), one can be concerned with favoring columns that help the IP master resolution.

Multiple v/s single and optimum v/s heuristic columns:

As part of the column selection strategy, one must decide on whether or not to solve the pricing subproblem optimally at every iteration, and whether to generate multiple columns at once or to solve the master after addition of a single column. For the MILS application, there is also the issue of selecting a column for each item or only treating one item per master iteration.

Although we have argued that minimum reduced cost columns are not necessarily best, the optimal subproblem solution may lead to a faster algorithm as reported for the binary cutting stock problem in Vance et. al. (1992) [44]. In general, our understanding, supported by what is reported in the literature (see for instance Degraeve (1992) [12]), is that, if the subproblem can be solved to optimality quite efficiently, then optimum/single column generation is superior. However, when the subproblem solution is computationally expensive, as is the case for the applications we consider, approximately-optimal/multiple column generation works better (see Section 5.5). The logic for adapting the strategy to the difficulty of the subproblem is to aim at balancing the computational workload between the master and the subproblem.

A simple column selection strategy:

To conclude this section, we describe the column selection strategy that we have implemented. After all these unfruitful tests, we have adopted a very simple scheme. At every iteration of the column generation algorithm, we first apply the approximation algorithm to solve the subproblem. If the best reduced cost of all heuristically generated columns is smaller than the a priori subproblem upper bounds (i.e. if the early termination conditions are not satisfied), we add all patterns that have a reduced cost equal to the best heuristic cost to the master formulation. Otherwise, we solve the subproblem optimally, and we add the optimum column to the master formulation if it is an admissible pattern (i.e. if its reduced cost is smaller than the a priori subproblem bounds).

4.6 Algorithm Overview

In this Section, we give an overall presentation of the algorithm. Figure 10 gives the basic flow of control of the computer code. We have previously described the specifics. Now we succinctly present the different stages through which the program passes: initialization, select node, process node, improve incumbent (i.e. getting integer solutions to the master), fathom node list, and print results. A description

of the code can be found in Appendix B.

4.6.1 Initialization

The program starts by reading the problem data and the parameter settings. Then the pool of columns is initialized with the application specific initial set of columns (see Section 4.2), plus artificial variables, plus eventually columns corresponding to an initial incumbent integer solution. The master and subproblem formulations are set up and loaded into the LP/MIP solver. The list of nodes in the branch-and-bound tree is initialized with the root node.

4.6.2 Select node

An iteration of the branch-and-bound algorithm consists of selecting a node to be processed, solving the node master linear relaxation by column generation in order to get a node lower bound, and, at some nodes, trying to improve the current incumbent integer solution. The order in which the nodes of the branch-and-bound tree are processed has a very significant influence on the overall performance of the algorithm.

We combine two standard node selection schemes: depth first search and best bound first search. Depth first search consists of processing next the descendant of the node that was just processed. Since it is more likely to find integer solutions down the tree, as the problem gets more restricted by the branching constraints, depth first search is applied first to hopefully lead to an incumbent integer solution that will help in pruning the tree. We consider that the incumbent solution is good enough once the gap between it and the current lower bound is smaller than a prespecified threshold. We then switch to best bound search in an effort to limit the number of nodes that will be processed. Indeed, the node with the best lower bound has to be processed anyway, so we might as well start with it. Other nodes with higher initial bounds may not have to be processed at all if we can get a new incumbent solution that prunes them. And the best bound node is also the most promising one as far as finding a better incumbent solution is concerned.

Together with the branching selection, the node selection scheme completely determines the sequence of the computations. In numerical experiments, we observed the great impact of these selection strategies on the tree size and the CPU time.

Figure 10: Algorithm flow of control.

However, what works best for one instance can be worse for another instance. So, we have no conclusive results to present.

4.6.3 Process node

Once a node is selected, we read the branching constraints that define the problem at the node; we detect basic inconsistencies (see Section 4.2); we generate a node specific initial set of columns (see Section 4.2); and we customize the master and the subproblem formulation. The algorithm proceeds by solving the master LP.

At every iteration of the column generation algorithm, we solve the restricted master LP problem. We also check if the LP solution happens to be integer. We collect the dual values. We solve the pricing subproblem (see Section 4.4). And we select columns to be added to the master formulation (see Section 4.5). Also, we test if column generation can be terminated using the conditions presented in Section 3.5.2. Note that these tests are performed twice at each master iteration: once after solving the master LP since $Z^u_{LP}(\tilde{Q})$ and Z^{INC} may have changed, and once after solving the subproblem since LB^u may have changed.

If the master LP solution contains artificial variables, their costs are increased and we return to column generation (see Section 4.2). Otherwise, we record the node bound, and we update the root lower bound accordingly. If the current node cannot be fathomed according to standard arguments, we choose how to branch (see Section 4.3). We define two successor nodes whose bound is initially set equal to the bound of their ancestor, and we add them to the list of unprocessed nodes.

4.6.4 Getting integer master solutions

One of the nice features of the column generation algorithm is that it is a primal approximation approach. The successive restricted master solutions are primal feasible and provide improving upper bounds on the optimum LP solution of $[M^u]$, but also on the optimum IP solution if the intermediate master solution happens to be integer. In practice, in the applications we consider, we often generate master integer solutions as a by product of the master LP optimization. In this way, we easily get incumbent solutions.

However, finding an optimal integer solution, i.e., closing the gap between the lower bound and the best integer solution cost, requires some work. For example, we solved an instance of the network design problem involving 9 nodes in 1:06 hours of CPU time. After 3 minutes of computation, the root node was solved and we already had the optimum lower bound of value 62 and an integer solution of cost 64. It took 1:03 hours and 86 more nodes to find an integer solution of cost 62.

The above example also illustrates the importance of getting tight cut-off values early in the procedure. If we had had an integer solution of cost 62 to start with, the solution time would have been 3 minutes instead of 1:06 hours. Thus, it is worth spending some computational effort in constructing integer solutions.

In this regard, we can exploit the information contained in the generated columns. By solving the restricted master integer program to optimality, we get the best integer solution that combines columns from the current set $\tilde{Q}$. We have observed that in general this integer solution is significantly better than those we obtain at intermediate stages of the LP optimization. For instance, in the above example, we obtained a primal integer solution of cost 94 during the LP optimization, while the best integer solution combining columns generated at the root node has cost 64. The time required to solve the restricted master by branch-and-bound is generally not prohibitive. On average, less than 10% of the total CPU time is spent in solving both the LP and IP restricted master formulations. The quality of the incumbent solutions obtained this way is fairly good, so we have not tried to generate integer solutions through other means such as a heuristic for the global problem.

This approach of solving the restricted master by standard branch-and-bound becomes computationally prohibitive when they are many alternate master solutions of the same cost. For instance, for the network design problem, if the edge demands are small relative to the ring capacity, there are many possible subset of edges that can form a feasible ring assignment. Then fixing a column selection variable λ, as is done in standard branch-and-bound, is not very restrictive, as there is a high probability of finding an alternate solution of the same cost that satisfies this branching constraint. Consequently, the branch-and-bound tree becomes quite large and the approach may fail to produce an optimum solution within the computational limits.

We observed that the time spent in solving the restricted master IP can increase, as the number of alternate master solutions gets larger, even taking from 50 to 80%

of the total CPU time. We have experimented with a restricted master IP approximation algorithm using a standard greedy heuristic for the covering problem (see Nemhauser and Wolsey (1988) [40]). However, the greedy solution of the restricted master IP is not as good as that obtained by branch-and-bound.

4.6.5 Algorithm termination

After a node has been processed, if a new primal integer solution has been found, we scan the list of unprocessed nodes to prune it. The algorithm terminates when this list is empty. We then print the results: the branch-and-bound tree, the statistics in terms of counters and times, and the best integer solution we have found.

5 Computational Results

We have implemented the algorithm in programming language C on a SUN work station SPARC 10 (Model 51). We use the CPLEX [9] callable library (version 2.1) as a linear and integer program solver. Initially, we used MINTO (Mixed INTeger Optimizer, [39]-[42]). MINTO is a software system built on top of CPLEX that solves mixed-integer linear programs by a branch-and-bound algorithm using linear programming relaxations. It provides automatic constraint classification, preprocessing, primal heuristics and constraint generation. It allows for column generation through the use of application routines that customize the code. Using MINTO, we quickly developed a code and gained some understanding of what are the important issues in the implementation. We then switched to our own implementation of branch-and-bound to have full control of every feature of the code. We opted for a very modular implementation that can quickly be adapted to different applications.

We have tested our algorithm on all four applications presented in Chapter 2: the network design problem (referred to as ND), the problem of network design with split assignment (NDSA), the clustering problem (CLUST), and the multi-item single-machine lot-sizing problem (MILS). We have performed extensive computational tests for the network design application in order to get some insights into a good implementation of IP column generation. Based on these, we have solved instances of the three other applications. For the network design applications (ND and NDSA), we use real data for practical size instances involving 5 to 21 nodes (i.e. 10 to 210 edges). We have also generated some instances by perturbing real data sets and modifying the ring capacity. For the two other applications, we use data from the literature. We now report the results of our computations on a variety of data sets.

5.1 The Network Design Problem

We have received real telecommunication traffic and network configuration data for the network design problem. Based on these, we have also generated some data sets by perturbing real traffic demands and modifying the ring capacity. We thus have a few test problems for our experimentation: some of which are real and others are generated. We refer to a specific instance by the number of nodes, followed by the ring capacity. We use a trailing letter to distinguish between multiple instances with the same number of nodes and capacity. The instances with less than 10 and more than 15 nodes represent complete networks. The instances with 10 to 15 nodes

represent incomplete networks. A sample of the perturbed test problems is given in Appendix C.

In Table 3, we report our results for all test problems for application ND. The option parameters of the program have been fixed to the setting that performs best on average across all instances. We switch from depth first search to best first search when the optimality gap is less than 30%. We do not use a cutting plane algorithm to solve the subproblem. We solve the restricted master IP by standard branch-and-bound when the node that has just been processed cannot be pruned and the number of columns that have been generated since the last call to branch-and-bound for the IP restricted master is at least twice the number of rows in the master (if only a few new columns have been generated, it is not worth resolving the master IP). The branching scheme consists of using cardinality branching if possible and otherwise branching on pairs of edges. We set an upper limit of 2 hours on the total CPU time. After 2 hours of computation, the algorithm stops generating columns and solves the restricted master IP by branch-and-bound to try to improve the incumbent solution before terminating.

Table 3 contains the following information: the problem name, the number of nodes in the master branch-and-bound tree, the total number of columns generated, the number of pricing subproblems that have been solved either heuristically or optimally, the number of times branch-and-bound has been used to solve the subproblem, the value of the master LP at the root node (i.e. the value of the restricted master upon node termination), the proven lower bound at the end of the algorithm, the value of the best integer solution found, and the total CPU time in terms of hours, minutes, and seconds (rounded down to the nearest integer).

The number of columns may exceed the number of pricing subproblems treated since at a given master iteration, we may generate multiple columns. Some subproblem solutions yield no new columns but simply prove LP optimality. So, the number of pricing subproblems solved by branch-and-bound represents partially a number of master LP optimality proofs, and partially a number of optimum reduced cost columns. The remaining columns have been generated with the heuristic.

Note also that the linear relaxation of the master provides a tight lower bound on the optimum integer solution. Across all instances solved, rounding the root LP bound up, and, in some cases, adding one to it, gives the value of the optimum

name	nod	col	SP	BB	rootLP	LB	UB	CPU time
7c50	1	47	32	3	22.15	23	23	0H:0M:3s
7c60	29	153	133	24	20.39	21	21	0H:0M:40s
7c60b	1	51	26	1	20.38	21	21	0H:0M:2s
7c100	3	77	71	11	14.97	16	16	0H:0M:15s
7c200	10	99	89	7	10.05	11	11	0H:0M:7s
7c240	15	384	352	103	9.58	10	10	0H:1M:47s
8c50	1	66	43	3	30.82	31	31	0H:0M:12s
8c60	39	116	88	17	30.91	32	32	0H:0M:47s
8c60b	7	79	58	11	31.88	33	33	0H:0M:35s
8c100	23	152	138	20	21.91	23	23	0H:0M:58s
8c200	9	230	222	38	14.83	16	16	0H:1M:35s
8c240	241	2744	2647	853	14.15	16	16	1H:22M:59s
9c50	6	76	50	4	42.77	43	43	0H:0M:35s
9c60	35	202	176	52	42.59	44	44	0H:5M:5s
9c60b	109	239	211	96	42.86	44	44	0H:11M:32s
9c100	69	380	331	97	33.03	34	34	0H:9M:23s
9c200	112	813	764	303	24.12	25	25	0H:44M:33s
9c240	58	619	582	309	21.70	23	23	0H:50M:16s
10c50	1	58	40	3	27.50	28	28	0H:0M:9s
10c60	9	122	110	29	24.70	26	26	0H:2M:0s
10c100	14	180	159	31	19.53	20	20	0H:1M:10s
10c200	16	255	225	70	13.93	15	15	0H:1M:10s
10c214	66	916	852	203	13.35	15	15	0H:20M:46s
12c50	1	86	31	1	72.33	73	73	0H:0M:19s
12c60	1	140	49	3	66.00	66	66	0H:0M:34s
12c100	213	551	581	216	51.13	53	53	0H:54M:48s
12c200*	402	732	612	181	35.46	37	39	2H:8M:41s
15c50	1	187	49	2	101.27	102	102	0H:1M:36s
15c60	1	193	103	4	88.60	89	89	0H:2M:37s
15c60b	12	181	69	4	94.53	95	95	0H:2M:4s
15c100	147	344	248	25	81.11	82	82	0H:13M:27s
18c100**	1	633	577	49	120.46	91	203	4H:28M:8s
21c50**	1	636	479	22	221.21	179	331	4H:7M:25s
21c60b**	1	645	478	30	215.75	162	323	4H:5M:11s

Table 3: Computational results for application ND.

name	master	subpr.	masLP	masIP	spHeur	spBB
7c50	7.2	90.1	4.2	2.9	5.9	80.7
7c60	3.6	95.1	2.5	1.0	2.3	91.0
7c60b	15.4	80.9	6.3	9.1	9.8	69.0
7c100	3.4	95.5	2.6	0.7	6.3	86.9
7c200	24.5	72.6	9.9	14.5	9.3	61.1
7c240	14.7	84.9	2.6	12.0	3.9	79.0
8c50	2.0	96.5	2.0	0.0	4.6	90.4
8c60	7.5	91.2	2.8	4.6	2.3	87.5
8c60b	1.9	97.3	1.6	0.3	2.0	94.1
8c100	4.9	94.1	3.0	1.8	4.8	87.8
8c200	11.7	87.8	3.3	8.4	7.6	78.5
8c240	71.7	28.1	1.0	70.7	1.2	26.1
9c50	1.8	97.5	1.6	0.2	3.0	93.7
9c60	1.0	98.7	0.7	0.2	1.2	96.6
9c60b	1.6	98.1	0.6	0.9	0.5	96.7
9c100	4.2	95.4	1.5	2.7	1.5	92.8
9c200	44.7	55.1	0.9	43.8	1.0	53.3
9c240	27.0	72.9	0.5	26.4	0.7	71.5
10c50	5.1	93.4	2.7	2.4	7.2	84.2
10c60	1.0	98.7	0.8	0.2	1.5	95.8
10c100	6.5	93.0	2.8	3.6	4.8	86.3
10c200	22.1	77.3	3.9	18.2	7.5	66.5
10c214	82.2	17.6	0.9	81.3	1.8	15.2
12c50	1.8	97.1	1.3	0.4	7.5	88.6
12c60	1.6	97.8	1.6	0.0	7.1	89.5
12c100	2.4	97.3	0.9	1.4	1.1	95.4
12c200*	64.4	35.4	1.4	62.9	0.8	34.2
15c50	0.8	98.9	0.8	0.0	6.1	92.3
15c60	1.8	97.9	1.1	0.6	9.6	87.8
15c214	3.4	96.0	1.7	1.6	7.3	88.0
15c100	6.3	92.6	3.5	2.7	5.0	86.9
18c100**	54.3	45.6	0.4	53.8	4.1	41.2
21c50**	48.7	51.2	0.4	48.3	5.9	45.1
21c60b**	51.4	48.5	0.4	50.9	6.0	42.3

Table 4: CPU time distribution in percent for application ND.

integer solution.

The problems become harder to solve as the size of the network (number of nodes) increases and, for the same size network, as the ring capacity increases. Recall that high ring capacity means a larger number of feasible solutions to the master IP. The larger problems are not solved to optimality within the allowed 2 hours of CPU time. By allowing 3 hours instead of 2 for these larger problems we do not get better results. However, many real-life problems involve less than 10 nodes. In comparison, a standard branch-and-bound approach applied to the natural integer formulation $[F_{ND}]$ fails to solve even the smaller problems (7 nodes) within reasonable time (we interrupted the resolution after 4 hours).

Note that as the column generation procedure at the root node may be terminated early (i.e. before LP optimality is proved), the root node LP value is not necessarily equal to the master LP value, but the root LP value rounded up is a valid lower bound on the optimal integer value. If the solution of an instance has been interrupted after 2 hours, we place a $*$ by its name. When this interruption occurs at the root node, we place a $**$ by the name of the instance. In that case, the root LP value information does not yield a valid bound and typically the gap between the best Lagrangian based lower bound and the best integer solution is still large.

In Table 4, we show where the algorithm spends its time. The first column of the table contains the problem name. Then, in the two following columns we give the percentage of the total CPU time that is spent solving respectively the master problem and the subproblem. We next decompose the restricted master solution time into LP optimization and IP optimization time (i.e. the time spent trying to improve the incumbent solution). For the subproblem, we decompose the time percentage into heuristic time and time spent in branch-and-bound. The given percentages do not add up to 100; the difference corresponds to the overhead.

Note that the most time consuming part of the algorithm is finding the exact solution of the pricing subproblem. However, as the ring capacity increases and the master solution space becomes larger, the time required to solve the master becomes significant. Be aware that the partition of the CPU time has a different interpretation when the solution of an instance has been interrupted at the root node (i.e. instances marked with $**$). Then, the portion of time spent on the subproblem is smaller partly because column generation has not yet passed the tailing off phase

which requires intensive computation to solve difficult subproblems.

For the network design problem with split assignments (NDSA), we use the same data sets as for ND. The demands are divided in two and each half (i.e. split demand) must be assigned to a different ring (this transformation is described in Section 2.2). These problems are harder to solve because, as a result of the splitting, there are more edge demands to consider for the same size of network. Moreover the granularity of demand relative to the capacity increases and there are some a priori edge disjunctive constraints. Given that the ring capacity is now larger relative to the average demand size, branching on pairs of edges often leads to an alternate solution of the same cost. Branching first on node cardinality leads to better performance on average. So, we try to eliminate a given fractional solution using cardinality branching, if it is not possible, we use node cardinality branching, and as a last resort, we use branching on pairs of edges.

We report our results for a few instances in Tables 5 and 6, using the same notation as before. One instance has been interrupted before 2 hours because we could not solve the subproblem to optimality within 10,000 branch-and-bound nodes. It is marked with a +.

5.2 The Clustering Problem

We have tested our algorithm on 3 types of clustering applications. These three classes of clustering problems have been studied by de Souza (1993) [13] and by Ferreira, Martin, de Souza, Weismantel and Wolsey (1994) [17]. We use the same data sets as these authors. The first type of instances arises from compiler design problems (see Johnson et al. (1993) [29]). Their test set consists of six problems and two possible cluster capacities. Each problem is referred to by a name starting with *cd* followed by *a* if the cluster capacity is 450 or a *b* if it is 512; next comes the number of nodes in the underlying incomplete graph. To distinguish between instances with the same number of nodes, we add a tailing *b* followed by the number of edges.

Johnson et al. have solved these instances on a RISC 6000 (Model 540) using a column generation algorithm and a cutting plane algorithm for the pricing subproblem. However, they only solved the root node master problem. Once LP optimality of the root node master is proven, they apply standard branch-and-bound to the

name	# nod	# col	# BB	rootLP	LB	UB	CPU time
6c50	12	95	15	20.11	21	21	0H:0M:28s
6c60	5	112	11	17.88	18	18	0H:0M:7s
6c100	6	91	13	14.43	15	15	0H:0M:6s
6c214	1	115	17	18.06	19	19	0H:0M:30s
7c50	35	411	128	27.91	29	29	0H:6M:49s
7c60	13	400	135	25.54	27	27	0H:11M:2s
7c60b	45	639	266	25.34	27	27	0H:15M:47s
7c100	24	576	147	19.99	21	21	0H:6M:6s
7c200	2	17	0	14.00	14	14	0H:0M:0s
7c240	2	15	0	14.00	14	14	0H:0M:0s
8c50	22	257	57	43.38	44	44	0H:6M:31s
8c60	53	476	228	39.67	41	41	0H:32M:15s
8c60b	115	662	404	40.22	42	42	1H:39M:27s
8c100	45	663	263	29.32	31	31	1H:50M:47s
8c200*	27	1240	295	20.26	22	23	2H:14M:44s
8c240*+	4	189	101	19.32	20	23	0H:20M:16s
9c50*	238	677	353	65.10	66	67	2H:0M:8s
9c60	130	545	261	63.87	65	65	1H:4M:56s
9c60b	87	639	352	61.25	62	62	1H:6M:45s
9c100*	56	880	422	47.21	48	49	2H:0M:7s
9c200*	27	709	219	32.00	33	37	2H:6M:2s
9c240*	16	914	343	29.61	30	34	2H:19M:28s
10c50	39	359	173	38.64	39	39	0H:22M:27s
10c60	20	293	61	34.34	35	35	0H:6M:54s
10c100*	72	1051	488	27.53	29	30	2H:1M:41s
10c200	2	13	0	20.00	20	20	0H:0M:0s
10c214	2	15	0	20.00	20	20	0H:0M:0s
12c50	120	416	62	100.91	102	102	0H:36M:52s
12c60*	156	615	210	88.02	89	90	2H:0M:13s
12c100*	17	564	95	70.70	71	120	2H:57M:40s
12c200*	5	536	78	48.63	49	102	2H:0M:21s
15c50*	49	458	74	159.76	160	161	2H:0M:57s
15c60*	184	555	53	141.27	142	143	2H:1M:19s
15c60b*	462	581	107	141.12	142	143	2H:0M:36s

Table 5: Computational results for application NDSA.

name	master	subpr.	masLP	masIP	spHeur	spBB
6c50	8.8	90.1	3.7	5.1	2.8	85.4
6c60	12.4	86.2	8.5	3.8	9.7	72.9
6c100	20.4	76.0	9.5	10.8	15.6	56.1
6c214	2.7	96.7	2.1	0.6	3.5	90.7
7c50	28.6	71.1	3.9	24.6	3.0	66.3
7c60	32.9	67.0	1.5	31.3	1.8	64.0
7c60b	25.9	73.9	3.0	22.8	2.0	70.4
7c100	41.5	58.1	6.0	35.4	4.9	51.3
7c200	13.6	40.9	13.6	0.0	36.3	0.0
7c240	9.0	36.3	9.0	0.0	36.3	0.0
8c50	26.7	73.0	1.6	25.1	2.1	69.9
8c60	29.2	70.6	1.2	28.0	1.0	68.7
8c60b	43.3	56.5	0.8	42.4	0.4	55.6
8c100	85.1	14.8	0.7	84.3	0.6	13.9
8c200*	93.3	6.5	1.4	91.9	1.1	5.1
8c240*+	1.2	98.7	0.5	0.6	0.7	82.3
9c50*	18.4	81.4	1.4	16.9	0.6	80.3
9c60	28.0	71.7	1.2	26.8	0.7	70.3
9c60b	13.5	86.3	1.1	12.4	0.8	84.6
9c100*	36.2	63.7	1.0	35.1	1.1	62.0
9c200*	63.6	36.3	0.8	62.7	1.1	34.9
9c240*	62.7	37.2	1.1	61.5	1.4	35.4
10c50	6.8	93.1	1.0	5.7	1.1	91.2
10c60	21.1	78.6	2.3	18.8	3.3	74.3
10c100*	78.0	21.9	1.6	76.4	0.8	20.7
10c200	7.6	50.0	7.6	0.0	46.1	0.0
10c214	14.2	53.5	14.2	0.0	53.5	0.0
12c50	17.7	82.0	2.0	15.6	1.9	79.5
12c60*	13.3	86.5	1.2	12.0	1.3	84.6
12c100*	61.9	38.0	0.2	61.6	0.9	36.8
12c200*	50.1	49.8	0.5	49.5	1.8	47.7
15c50*	4.4	95.4	0.4	4.0	1.7	93.4
15c60*	29.5	70.1	1.9	27.5	2.1	67.7
15c60b*	12.8	86.4	3.4	9.4	1.8	84.2

Table 6: CPU time distribution in percent for application NDSA.

restricted master to get a good integer solution. If there is an optimality gap (which only happens in 3 out of 12 instances) they do not pursue the search for an exact solution of the master. The CPU times we report in Table 7 improve upon theirs by a factor of between 2 and 20 depending on the instance.

The second type of test problems arises in VLSI placement design (see Junger et. al. (1992) [30]). These test problems have a name starting with *vlsi* followed by the number of nodes in the graph.

The third class of test problems are clustering problems arising in finite element computations (de Souza (1994) [14]). They consist of finding an equipartition of a sparse graph that minimizes the number of edges in the cut. We refer to them with a name starting by *mesh* and followed by the number of nodes in the graph. Tables 7 and 8 contain the computational results for the 3 classes of test problems. The parameter settings leading to the best results on average are: a switch from depth first to best first search for a gap of 20%, no cutting planes for the subproblem, and branching on pairs of nodes only. After 2 hours of CPU time, we stop generating new columns and we solve the restricted master IP by branch-and-bound, hoping to improve our incumbent solution. By allowing 3 hours we do not improve our results except for problems $vlsi42$ and $vlsi166$. We also report computation results with a 3 hour limit for instances $vlsi42$ and $vlsi166$, marked with a $h3$.

Ferreira et. al. (1994) [17] have developed a branch-and-cut approach for the clustering problem. It is interesting to compare the two approaches: IP column generation and branch-and-cut. In Table 9, we present comparative results. In the columns of this table we give the problem names, the root node upper bound value obtained with our IP Column Generation algorithm ($CGrootBd$), the best upper bound that is obtained at the root node when using the Branch-and-Cut approach ($BCrootBd$), and the value of the optimal solution (when it is not known, we give our best lower bound and we place a $*$ by the value). Then, in the column entitled $BCtime$ (resp. $CGtime$), we report the total computational time (on the same machine) given in [17] for the branch-and-cut approach (resp. obtained with our algorithm). We place a $*$ when the instance has not been solved to optimality.

Comparing CPU times, we note that the instances that are difficult for the branch-and-cut method are the same as the difficult ones for the column generation method. Second, the solution times are of the same order of magnitude with both approaches

name	nd	col	BB	rootLP	UB	LB	time
cda30b47	1	33	3	1099.00	1099	1099	0H:0M:4s
cda30b56	1	32	4	1642.00	1642	1642	0H:0M:6s
cda45	1	63	7	2928.00	2928	2928	0H:0M:55s
cda47b99	1	66	11	1837.00	1837	1837	0H:1M:53s
cda47b101	5	128	39	3574.00	3569	3569	0H:7M:23s
cda61	7	264	96	22245.00	22216	22216	1H:26M:24s
cdb30b47	1	13	5	1174.00	1174	1174	0H:0M:5s
cdb30b56	1	35	3	1748.00	1748	1748	0H:0M:6s
cdb45	1	53	14	3238.00	3238	3238	0H:1M:17s
cdb47b99	1	77	18	1993.00	1993	1993	0H:3M:37s
cdb47b101	3	155	46	3969.00	3960	3960	0H:7M:18s
cdb61	1	91	15	23564.00	23564	23564	0H:17M:9s
vlsi15	4	40	14	96.00	95	95	0H:0M:4s
vlsi17	3	49	12	48.00	47	47	0H:0M:37s
vlsi34	2	202	47	183.00	183	183	0H:2M:20s
vlsi37	1	143	35	211.50	211	211	0H:1M:51s
vlsi38	17	1345	669	285.80	282	282	1H:8M:44s
vlsi42*	6	1093	403	408.00	407	406	2H:0M:0s
vlsi42$h3$	7	1385	516	408.00	406	406	2H:14M:10s
vlsi43	7	660	216	526.00	524	524	0H:9M:46s
vlsi44	7	905	388	527.00	524	524	0H:18M:23s
vlsi46	1	381	169	491.00	491	491	0H:10M:6s
vlsi48	1	414	163	522.00	522	522	0H:9M:49s
vlsi166**	1	2221	51	2056.86	2402	1909	2H:48M:25s
vlsi166*$h3$	1	2549	119	2073.81	2377	1969	3H:25M:15s
mesh31	2	203	82	44.00	44	44	0H:0M:58s
mesh70	1	19	1	113.00	113	113	0H:0M:3s
mesh138**	1	1648	949	214.68	232	170	2H:11M:23s
mesh148**	1	535	0	246.40	265	244	2H:2M:6s
mesh274**	1	643	83	447.02	469	437	2H:0M:26s

Table 7: Computational results for application CLUST.

name	master	subpr.	masLP	masIP	spHeur	spBB
cda30b47	1.0	95.9	1.0	0.0	1.6	89.2
cda30b56	0.7	95.0	0.7	0.0	2.0	87.9
cda45	0.2	99.2	0.2	0.0	1.0	95.3
cda47b99	0.2	99.5	0.2	0.0	0.5	96.8
cda47b101	0.2	99.6	0.1	0.0	0.3	97.0
cda61	0.0	99.8	0.0	0.0	0.1	98.2
cdb30b47	1.1	95.5	1.1	0.0	1.9	88.5
cdb30b56	0.9	96.1	0.9	0.0	1.9	90.6
cdb45	0.3	99.3	0.3	0.0	0.9	94.6
cdb47b99	0.1	99.7	0.1	0.0	0.4	97.3
cdb47b101	0.2	99.6	0.2	0.0	0.4	96.6
cdb61	0.0	99.8	0.0	0.0	0.3	98.4
vlsi15	4.8	89.5	4.4	0.3	0.7	77.2
vlsi17	0.5	99.0	0.4	0.0	0.1	98.6
vlsi34	1.5	98.3	0.8	0.6	0.9	92.7
vlsi37	0.9	98.7	0.9	0.0	1.4	89.9
vlsi38	3.0	96.9	0.6	2.4	0.3	86.8
vlsi42$*$	0.6	99.3	0.5	0.0	0.2	94.8
vlsi42$h3$	0.7	99.2	0.7	0.0	0.3	98.8
vlsi43	2.1	97.7	2.0	0.0	1.6	86.8
vlsi44	2.1	97.7	2.0	0.1	1.2	87.6
vlsi46	1.6	98.3	1.6	0.0	1.1	92.3
vlsi48	1.9	97.9	1.9	0.0	1.4	91.2
vlsi166$**$	49.4	50.5	20.6	28.7	27.9	19.7
mesh31	5.5	94.0	5.4	0.1	2.1	78.9
mesh70	0.4	91.5	0.4	0.0	32.7	48.3
mesh138$**$	43.6	56.3	35.0	8.6	5.1	33.6
mesh148$**$	88.4	11.5	87.1	1.3	11.5	0.0
mesh274$**$	45.3	54.5	45.2	0.1	40.1	7.6

Table 8: CPU time distribution in percent for application CLUST.

name	CGrootBd	BCrootBd	optimum	CG time	BC time
cda30b47	1099	1099	1099	0H:0M:4s	0H:0M:26s
cda30b56	1642	1642	1642	0H:0M:6s	0H:0M:16s
cda45	2928	2966	2928	0H:0M:55s	0H:20M:44s
cda47b99	1837	1848	1837	0H:1M:53s	0H:18M:43s
cda47b101	3574	3682	3569	0H:7M:23s	2H:43M:25s
cda61	22245	na	22216	1H:26M:24s	27H*
cdb30b47	1174	1174	1174	0H:0M:5s	0H:0M:21s
cdb30b56	1748	1748	1748	0H:0M:6s	0H:0M:24s
cdb45	3238	3259	3238	0H:1M:17s	0H:17M:04s
cdb47b99	1993	2006	1993	0H:3M:37s	0H:56M:50s
cdb47b101	3969	4034	3960	0H:7M:18s	0H:26M:09s
cdb61	23564	23577	23564	0H:17M:9s	2H:46M:47s
vlsi15	96	96	95	0H:0M:4s	0H:0M:8s
vlsi17	48	50	47	0H:0M:37s	0H:17M:7s
vlsi34	183	184	183	0H:2M:20s	0H:1M:33s
vlsi37	211	212	211	0H:1M:51s	0H:1M:39s
vlsi38	285	287	282	1H:8M:44s	2H:30M:58s*
vlsi42	408	407	406	2H:14M:10s	0H:20M:10s
vlsi44	527	525	524	0H:18M:23s	0H:2M:0s
vlsi46	491	495	491	0H:10M:6s	0H:59M:15s
vlsi48	522	528	522	0H:9M:49s	5H:26M:20s
vlsi166	2377	2428	*1969	3H:25M:15s**	5H:29M*
mesh31	44	44	44	0H:0M:58s	0H:0M:5s
mesh70	113	113	113	0H:0M:3s	0H:1M:43s
mesh138	232	224	224	2H:11M:23s**	1H:48M:36s
mesh148	265	258	258	2H:2M:6s**	0H:14M:11s
mesh274	469	462	462	2H:0M:26s**	1H:32M:02s

Table 9: Comparing our results for CLUST with branch-and-cuts results.

with some variations depending on the problem type. For the compiler design problems IP column generation is significantly faster. For the VLSI problems, the advantage depends on the instance. For the finite element problems, our algorithm fails to solve the larger instances while the branch-and-cut approach succeeds. For these *mesh* problems, the master formulation is highly degenerate: the size of the basis is greater than the number of nodes, while in an optimal IP solution, only 2 columns have a non zero value. We observe that the master LP solution requires 35 to 87 % of the time spent trying to solve these large instances.

5.3 The Multi-item Single-machine Lot-sizing Problem

Constantino (1994) [7] has developed a branch-and-cut algorithm for the multi-item single-machine lot-sizing problem. Most of the cuts he uses concern the subproblem. We have integrated his separation subroutines in the cutting plane algorithm for the pricing subproblem. We have tested IP column generation on his set of test problems.

The name of an instance is of the type $mbnvicjsk$, where m is the number of item, n is the number of period, i refers to a set of demands for this size problem, j refers to the machine capacity, and k refers to the start-up cost. For a detailed description of these data sets, the reader is referred to Constantino (1994) [7].

The algorithm settings we use for this problem consist of a 20% gap for the switch to best first search, and branching on a pair consisting of a period and an item. We have not implemented any heuristic procedure to solve the subproblem. Instead, we solve it exactly at every master iteration, using cutting planes followed by branch-and-bound. Our results are presented in Tables 10 and 11. In Table 10, we have added a column to report the total number of cuts we have generated for the subproblem during the cutting plane algorithm we use at the root node of the subproblem branch-and-bound tree. We manage the cuts using a pool and remove them from the subproblem formulation after each master iteration. In Table 11, we show the portion of time spent on the subproblem solution using respectively the cutting plane algorithm and branch-and-bound applied to the strengthened subproblem formulation.

The comparison with the branch-and-cut approach used by Constantino shows that our CPU times are worse (by a factor of about 1 to 3). We have not tested how the two approaches compare on larger instances; nor have we implemented a heuristic

name	nod	col	BB	cuts	rootLP	LB = UB	CPU time
5b12v1c12s2	1	75	13	1219	3479846.00	3479846	0H:0M:13s
5b12v2c12s2	1	57	0	763	873644.00	873644	0H:0M:5s
5b12v3c12s2	1	36	0	511	888844.00	888844	0H:0M:4s
5b24v1c12s2	1	111	24	3600	1723945.00	1723945	0H:1M:41s
5b24v2c12s2	1	52	3	1375	1811497.00	1811497	0H:0M:24s
5b24v3c12s2	1	79	3	2051	1255917.00	1255917	0H:0M:41s
5b36v1c12s2	1	161	48	8246	947394.00	947394	0H:8M:24s
5b36v2c12s2	1	38	3	2023	867049.00	867049	0H:1M:9s
5b36v3c12s2	43	481	435	4918	1526633.00	1529170	0H:41M:26s
5b12v1c16s2	1	63	4	856	2278268.00	2278268	0H:0M:8s
5b12v2c16s2	1	39	0	455	718781.00	718781	0H:0M:2s
5b12v3c16s2	1	24	0	316	740190.00	740190	0H:0M:1s
5b24v1c16s2	3	98	15	2509	1506855.50	1507041	0H:1M:9s
5b24v2c16s2	1	40	0	1093	1785755.00	1785755	0H:0M:20s
5b24v3c16s2	1	45	0	1233	1095568.00	1095568	0H:0M:19s
5b36v1c16s2	1	63	15	2640	724301.00	724301	0H:1M:46s
5b36v2c16s2	1	26	0	1663	834056.00	834056	0H:0M:37s
5b36v3c16s2	7	141	8	2466	1377965.00	1378455	0H:4M:46s
5b12v1c24s2	1	30	3	427	1893442.00	1893442	0H:0M:2s
5b12v2c24s2	1	32	0	350	694991.00	694991	0H:0M:1s
5b12v3c24s2	1	24	0	341	737190.00	737190	0H:0M:1s
5b24v1c24s2	1	56	2	1074	1273593.00	1273593	0H:0M:18s
5b24v2c24s2	1	35	0	898	1764143.00	1764143	0H:0M:8s
5b24v3c24s2	1	27	0	1005	1091448.00	1091448	0H:0M:11s
5b36v1c24s2	3	35	9	1560	652187.50	652634	0H:1M:1s
5b36v2c24s2	1	27	0	1531	818018.00	818018	0H:0M:31s
5b36v3c24s2	39	210	0	2121	1367520.00	1368279	0H:7M:20s
5b12v1c12s4	1	75	19	1024	3679846.00	3679846	0H:0M:10s
5b12v2c12s4	1	57	1	842	1053644.00	1053644	0H:0M:7s
5b12v3c12s4	1	43	0	667	1068844.00	1068844	0H:0M:4s
5b24v1c12s4	1	110	7	4273	1998430.00	1998430	0H:2M:0s
5b24v2c12s4	1	48	4	1138	1986964.00	1986964	0H:0M:14s
5b24v3c12s4	9	185	102	2938	1521427.43	1522772	0H:4M:29s
5b36v1c12s4	1	155	77	7007	1268385.00	1268385	0H:11M:50s
5b36v2c12s4	1	43	7	1711	1101279.00	1101279	0H:1M:11s
5b36v3c12s4	9	236	75	4043	1872207.67	1876742	0H:12M:5s
5b12v1c16s4	1	55	5	806	2478268.00	2478268	0H:0M:7s
5b12v2c16s4	1	39	0	374	858781.00	858781	0H:0M:2s
5b12v3c16s4	1	27	2	403	860976.00	860976	0H:0M:1s

Table 10: Computational results for application MILS.

name	master	subpr.	masLP	masIP	spCut	spBB
5b12v1c12s2	0.8	98.2	0.8	0.0	82.1	13.6
5b12v2c12s2	1.1	96.5	1.1	0.0	94.7	0.0
5b12v3c12s2	0.7	95.3	0.7	0.0	92.4	0.0
5b24v1c12s2	0.2	99.5	0.2	0.0	79.5	19.7
5b24v2c12s2	0.2	98.8	0.2	0.0	94.2	3.9
5b24v3c12s2	0.2	99.2	0.2	0.0	94.6	3.8
5b36v1c12s2	0.0	99.8	0.0	0.0	68.8	30.5
5b36v2c12s2	0.1	99.5	0.1	0.0	84.5	14.3
5b36v3c12s2	0.3	99.6	0.2	0.1	68.4	30.5
5b12v1c16s2	0.9	97.3	0.9	0.0	92.3	2.9
5b12v2c16s2	0.0	94.2	0.0	0.0	92.2	0.0
5b12v3c16s2	1.9	90.3	1.9	0.0	86.5	0.0
5b24v1c16s2	0.2	99.4	0.2	0.0	88.2	10.7
5b24v2c16s2	0.3	98.9	0.3	0.0	97.8	0.0
5b24v3c16s2	0.2	98.6	0.2	0.0	97.3	0.0
5b36v1c16s2	0.1	99.7	0.1	0.0	78.2	20.8
5b36v2c16s2	0.0	99.3	0.0	0.0	98.3	0.0
5b36v3c16s2	0.2	99.6	0.2	0.0	97.2	1.4
5b12v1c24s2	1.2	90.9	1.2	0.0	86.3	2.5
5b12v2c24s2	3.0	90.7	3.0	0.0	83.4	0.0
5b12v3c24s2	0.9	90.0	0.9	0.0	86.3	0.0
5b24v1c24s2	0.1	98.8	0.1	0.0	96.3	1.1
5b24v2c24s2	0.1	98.0	0.1	0.0	97.1	0.0
5b24v3c24s2	0.6	97.7	0.6	0.0	96.2	0.0
5b36v1c24s2	0.0	99.4	0.0	0.0	88.9	9.4
5b36v2c24s2	0.2	99.1	0.2	0.0	98.5	0.0
5b36v3c24s2	0.2	99.5	0.2	0.0	98.5	0.0
5b12v1c12s4	1.2	97.1	1.2	0.0	83.7	12.6
5b12v2c12s4	0.7	97.1	0.7	0.0	93.1	1.4
5b12v3c12s4	1.1	95.8	1.1	0.0	93.5	0.0
5b24v1c12s4	0.1	99.7	0.1	0.0	95.8	3.5
5b24v2c12s4	0.2	98.1	0.2	0.0	94.5	2.5
5b24v3c12s4	0.3	99.4	0.3	0.0	71.4	27.3
5b36v1c12s4	0.0	99.8	0.0	0.0	38.5	61.0
5b36v2c12s4	0.1	99.4	0.1	0.0	81.6	17.0
5b36v3c12s4	0.2	99.6	0.2	0.0	65.5	33.5
5b12v1c16s4	0.9	96.6	0.9	0.0	89.6	4.7
5b12v2c16s4	0.7	94.2	0.7	0.0	91.9	0.0
5b12v3c16s4	2.5	89.9	2.5	0.0	78.9	5.8

Table 11: CPU time distribution in percent for application MILS.

or a dynamic programming algorithm for the subproblem.

5.4 Early Termination of Column Generation

In this Section, we want to test the hypothesis that the early termination conditions presented in Section 3.5.2 help to reduce the amount of computation. We compare the performance of the IP column generation algorithm with and without the use of the conditions for early termination of the column generation procedure (cfr Section 3.5.2) and the resulting a priori upper bounds on the subproblem value (cfr Section 3.5.3).

With the IP column generation algorithm as with other IP algorithms, comparisons between different options are not always easy because there is a factor of chance in the performance of the algorithm. If, at one stage of the procedure, one generates a column, say k, rather than another column, say l, the rest of the procedure may take a completely different path of actions. For instance, column k and l may yield different dual solutions and, in turn, different new columns. Also, column k may be the right complement to existing columns in forming an integer solution, while l may not be. In the case of this test, by stopping column generation early, we may not generate columns that would have been very helpful and by setting an upper cut-off for the subproblem branch-and-bound procedure, we may get a different subproblem solution. Consequently, it is important to run a test on a large set of instances to see what is the average behavior.

In addition, comparing two versions of the algorithm on the same instance, we might observe that some performance parameters (number of columns, number of subproblem solutions, number of calls to subproblem branch-and-bound, etc ...) are better while others are worse. The total CPU time summarizes these sometimes contradictory effects.

In Table 12, we present the total CPU time (in seconds) needed to solve the network design instances with four different versions of the IP column generation algorithm. Version $ETCaSPB$ corresponds to the algorithm settings presented in Section 5.1, in which we use the conditions for early termination of the column generation procedure as well as the resulting a priori upper bounds on the subproblem value. It is our benchmark in the present comparative test. In Version $noSPB$, all algorithmic options are equal to those of $ETCaSPB$, but we do not use the a priori bound on

the subproblem value presented in Proposition 6. In Version $noETC$, we go one step further: we do not use the column generation early termination condition (ii), $\lceil LB^u \rceil \geq Z^u_{LP}(\tilde{Q})$, nor condition (iii), $\lceil LB^u \rceil \geq Z^{INC}$ (cfr the end of Section 3.5.2). In Version $noETCii$, we only use only condition (i), $v^u(\pi, \mu, \nu) \geq 0$, and condition (iii), $\lceil LB^u \rceil \geq Z^{INC}$, to terminate column generation at a node, but we do not use condition (ii), $\lceil LB^u \rceil \geq Z^u_{LP}(\tilde{Q})$. The last line of the table contains the sum of the CPU times for all instances for a each algorithm version.

Looking at the times presented in Table 12, we conclude that the early termination of column generation helps in solving problems faster. Comparing $ETCaSPB$ and $noSPB$, we note that the use of the a priori subproblem bounds improves the performance in 24 out of 30 instances and the speed up factor varies from around 0.5 (twice as slow) to 14.5. Comparing $ETCaSPB$ and $noETC$, we see that the combined use of early termination conditions and a priori subproblem bounds improves the performance in 29 out of 30 instances and the speed up factor varies from around 0.7 to 20. Comparing $noETCii$ and $noETC$ shows that early termination condition (ii) is the main contributor to the time reduction in the case of the network design application.

5.5 Comparing Some Implementation Strategies

In this section, we want to illustrate the impact of some of the implementation choices, which we discussed in Chapter 4, on the performance of our algorithm. As in the comparative tests of the previous Section, we use the total CPU time measure to summarize the effect of any particular implementation option.

Table 13 contains the computation time obtained for the network design (ND) problems with the different versions of the algorithm that we want to compare. The last line of the table contains the average CPU time across all instances for a each algorithm version. The first column contain the problem names. The headings of the other columns refer to the algorithm versions that we have tested. $BENCH$ stands for the benchmark version presented in Section 5.1. To define other versions, we mention in what way they differ from the benchmark.

Version $FULLP$ solves the LP programs using full pricing instead of the default CPLEX option of partial pricing. Bixby, the developer of CPLEX, has suggested using the full pricing option when using a column generation procedure. We observe a significant time reduction which cannot be explained by reduction in master

name	ETCaSPB	noSPB	noETCii	noETC
7c50	3.9	4.6	4.2	4.4
7c60	41.0	42.1	46.4	48.5
7c60b	2.2	2.7	2.8	2.8
7c100	14.2	8.8	13.7	14.4
7c200	7.2	36.0	56.3	58.9
7c240	109.0	86.0	200.3	257.4
8c50	11.6	14.8	14.1	14.7
8c60	48.0	54.2	160.8	168.5
8c60b	36.1	42.1	47.7	51.0
8c100	61.9	68.5	1240.7	1297.8
8c200	100.2	95.2	140.3	152.2
8c240	4924.5	5926.8	4762.2	5246.7
9c50	35.6	42.3	300.5	324.8
9c60	262.1	292.6	258.3	276.2
9c60b	615.2	701.4	1109.6	1056.1
9c100	560.2	746.2	2338.1	2378.5
9c200	2645.8	1187.1	1842.5	1941.4
9c240	2957.3	3195.1	5476.2	5599.8
10c50	8.9	9.8	9.2	9.4
10c60	114.7	151.3	137.3	169.4
10c100	69.8	29.8	230.5	238.3
10c200	69.2	1007.8	4326.1	1367.3
10c214	1204.2	725.9	1287.5	1283.2
12c50	18.9	24.5	23.6	23.4
12c60	34.6	39.5	38.9	41.0
12c100	3271.8	3679.2	4713.1	5240.4
15c50	97.0	111.6	108.3	111.3
15c60	159.6	209.4	210.0	208.2
15c60b	124.8	140.7	970.2	1069.4
15c100	790.9	735.8	4039.7	3924.7
TOTAL	18400.6	19411.7	34108.9	32580.1

Table 12: Early termination of column generation: time comparison for ND.

LP solution time alone. In fact, for the instances where the benchmark algorithm spends significant time in solving the master LP (more than 7% for instances 7c200, 10c200 and 10c240 (see Table 4)), the full pricing version is slower. One possible explanation is that the better performance of full pricing is due to better dual master solutions yielding different column generation patterns (paths).

Version $PART$ uses full pricing and also uses a partitioning master formulation rather than a covering formulation. Since we have not developed a procedure to deal with master infeasibilities in the case of a partitioning formulation, we had to interrupt the solution, for 4 instances (marked by inf) out of 28, due to master LP infeasibilities. Comparing with the $FULLP$ version, we observe a significant increase in CPU time.

In version IOC, we have initialized the IP column generation algorithm with an optimal IP solution, including the columns of the optimal solution in the initial master formulation. Contrary to the conclusions of our previous research [45] (see also Section 4.5), we observe that IOC yields a significant reduction in the amount of computation. However, in [45], we were using column generation at the root node only, and we did not use any column generation early termination criteria. So, one possible interpretation, which explains this apparent contradiction, is that the computational reduction we observe for the network design application is due to the fact we have a priori the value of the optimal solution rather than due to the presence of the optimal columns in the initial formulation. This cut-off value serves not only to prune the master branch-and-bound tree, but also to terminate column generation early at each node of the branch-and-bound tree.

To confirm this interpretation, we have tested yet another version, IOV, in which we initialize the IP column generation algorithm with the optimal IP value, but we do not include the corresponding columns in the initial master formulation. As shown in Table 13, the average computation time improves a little over the IOC version, supporting the above interpretation.

In version SOC, we experiment with another column generation strategy consisting of generating one optimal reduced cost column at each iteration of the column generation procedure. The CPU time increases significantly as a result. This test shows that it is preferable to use a multiple/heuristic column generation strategy as discussed in Section 4.5.

name	BENCH	FULLP	PART	IOC	IOV	SOC
7c50	3.9	4.8	4.6	4.1	3.3	24.6
7c60	41.0	16.3	34.2	7.1	7.8	67.2
7c60b	2.2	2.5	10.3	1.9	2.8	44.6
7c100	14.2	10.1	16.0	5.5	9.4	50.8
7c200	7.2	19.0	inf	10.3	8.9	17.0
7c240	109.0	70.2	76.3	0.6	0.6	72.3
8c50	11.6	6.3	10.4	11.5	11.1	88.6
8c60	48.0	40.0	29.6	49.0	49.0	131.3
8c60b	36.1	46.4	37.3	58.1	50.0	142.9
8c100	61.9	151.7	189.9	41.3	33.3	1178.8
8c200	100.2	110.1	inf	94.1	94.9	265.7
8c240	4924.5	3980.9	uns	5063.9	4589.1	5636.7
9c50	35.6	51.9	79.6	29.7	42.1	357.2
9c60	262.1	242.9	251.2	237.9	223.1	370.4
9c60b	615.2	416.4	626.8	840.6	462.5	1022.8
9c100	560.2	992.5	572.8	99.2	102.8	2415.7
9c200	2645.8	792.3	4478.5	101.2	106.2	3946.9
9c240	2957.3	1338.4	6366.2	171.1	170.3	uns
10c50	8.9	11.6	24.3	3.8	4.0	53.0
10c60	114.7	75.7	82.6	84.8	76.8	198.2
10c100	69.8	89.0	55.1	24.5	22.5	150.0
10c200	69.2	233.1	inf	45.8	56.4	2248.8
10c214	1204.2	1913.9	inf	320.6	344.3	733.2
12c50	18.9	19.5	18.5	17.6	30.9	297.5
12c60	34.6	58.5	18.8	18.0	17.9	579.2
12c100	3271.8	2117.3	2388.4	2839.5	2714.7	6654.6
15c50	97.0	82.5	50.5	77.2	79.2	2155.6
15c60	159.6	148.2	147.7	61.4	116.2	2166.3
15c60b	124.8	67.0	102.6	62.5	52.3	1729.8
AVERAGE	607	452	653	358	327	1171

Table 13: Comparing CPU time required to solve ND problems with different implementation options (*uns* stands for unsolved within the allocated 2 hours).

5.6 Approximation Algorithm

As illustrated in the previous tables, we cannot solve the larger instances to optimality within reasonable time using our implementation of the IP column generation algorithm. Limiting the total CPU time, we have produced an approximately optimum solution and a bound to attest to its quality. However, there are other ways to truncate the exact solution procedure in order to obtain approximate solutions. One may set an upper limit on counters like the number of columns, the number of optimally generated columns, the number of master iterations, or the number of nodes in the branch-and-bound tree.

Another way to truncate the search tree is to set an a priori cut-off value. This amounts to searching if there exists an integer solution that has a cost lower than or equal to the a priori cut-off value. However, if the optimal value is equal to the cut-off value, the algorithm will not necessarily provide such a solution. Our experience with the network design application is that the size of the branch-and-bound tree is very much dependent on a tight cut-off value early in the procedure. So for these problems, checking the existence of a solution of given cost can be much faster than solving the optimization problem.

A third way to limit the computations consists of searching for an integer solution whose cost is within α % of optimality. One may specify a tolerance factor (TOL) in absolute value. Considering the minimization problem $[M]$, we obtain an absolute tolerance from the percentage specification by letting $TOL = (100 + \alpha)/100 * LB$ where LB is the best known lower bound. Then, the criteria for pruning a branch-and-bound node by bound becomes

$$LB^u + TOL \geq Z^{INC}$$

To take advantage of the tolerance factor for early termination of column generation, one can use $LB^u + TOL$ instead of LB^u in the early termination conditions presented in Section 3.5.2 as well as in the a priori subproblem bound computations. For maximization problems the argument is similar.

We have tested the use of a tolerance factor for the network design problem with and without split assignments. The strategy that seems to work best consists of introducing a tolerance after the completion of the root node solution. In Tables

14-17, we present the computational results obtained for large instances using an absolute tolerance of 1 unit.

Comparing these results with the 2 hours limit truncation (Tables 3-6), we note that for all instances (but one) we get the same incumbent value (or even a better value), sometimes much quicker, but the lower bound is obviously not always as good. Also the portion of time spent in solving the subproblems decreases since, by using a tolerance, we truncate the cumbersome master LP optimality proofs.

name	# nod	# col	# BB	rootLP	LB	UB	CPU time
8c200	57	495	5	14.83	15	16	0H:1M:31s
8c240	3	66	12	14.15	15	16	0H:0M:21s
9c200	186	525	17	24.12	25	26	0H:26M:11s
9c240	65	666	276	21.70	22	23	0H:28M:59s
10c200	50	992	76	13.93	14	15	0H:5M:31s
10c214	56	640	69	13.35	14	15	0H:3M:16s
12c200*	193	817	98	35.46	36	39	2H:1M:12s

Table 14: Computational results for application ND with TOL.

name	master	subpr.	masLP	masIP	spHeur	spBB
8c200	82.4	16.7	9.9	72.4	11.4	4.5
8c240	6.3	92.4	3.3	3.0	7.4	82.4
9c200	93.6	6.1	1.4	92.1	0.9	5.1
9c240	46.8	53.0	1.1	45.7	1.2	50.8
10c200	84.8	14.8	3.6	81.2	6.3	7.6
10c214	82.9	16.6	2.8	80.1	3.7	11.4
12c200*	88.6	11.2	0.8	87.7	0.8	10.2

Table 15: CPU time distribution in percent for application ND with TOL.

name	# nod	# col	# BB	rootLP	LB	UB	CPU time
8c200*	37	1612	269	20.26	21	22	2H:4M:6s
8c240	32	1833	429	19.32	20	21	0H:57M:10s
9c50	3	275	39	65.10	66	67	0H:11M:47s
9c100	73	551	112	47.21	48	49	1H:6M:57s
9c200*	33	737	125	32.00	32	37	2H:0M:2s
9c240*	23	951	278	29.61	30	34	2H:5M:9s
10c100*	68	1233	394	27.53	28	30	2H:8M:58s
12c60	3	334	33	88.02	89	90	0H:19M:29s
12c100*	23	570	71	70.70	71	120	2H:45M:5s
12c200*	16	592	56	48.63	49	102	3H:3M:52s
15c50	3	348	22	159.76	160	161	0H:34M:35s
15c60	3	466	22	141.27	142	143	1H:8M:27s
15c60b	3	404	25	141.12	142	143	0H:35M:48s

Table 16: Computational results for application NDSA with TOL.

name	master	subpr.	masLP	masIP	spHeur	spBB
8c200*	94.7	5.1	1.3	93.4	1.4	3.4
8c240	78.5	21.3	6.4	72.0	4.6	15.8
9c50	21.7	78.1	0.6	21.1	2.0	75.5
9c100	70.6	29.2	1.2	69.4	1.0	27.8
9c200*	85.2	14.7	1.0	84.1	1.1	13.3
9c240*	75.8	24.1	1.3	74.4	1.6	22.1
10c100*	86.8	13.0	1.5	85.3	1.1	11.6
12c60	21.8	78.0	0.7	21.1	4.0	73.4
12c100*	67.5	31.6	0.3	67.2	1.0	30.4
12c200*	81.7	18.2	0.5	81.2	1.2	16.8
15c50	14.1	85.8	0.4	13.6	4.1	81.3
15c60	49.6	50.3	0.3	49.2	2.9	47.1
15c60b	32.0	67.9	0.4	31.6	3.5	64.0

Table 17: CPU time distribution in percent for application NDSA with TOL.

6 Conclusions

Summary

This work shows that the decomposition of an integer program can provide a reformulation whose LP relaxation yields tight bounds. This in turn allows us to solve difficult integer programs to optimality. However, dealing with an integer program that has an enormous (implicit) number of columns requires the integration of a column generation procedure in the exact solution algorithm. The standard IP techniques must be adapted to become compatible with column generation.

We have proposed a branching scheme that offers a unified view of branching rules previously used in a column generation framework and extends the class of problems that can be dealt with. We also have derived conditions for early termination of the column generation procedure. In the process of developing an implementation of the IP column generation algorithm, we have discussed a few algorithmic choices that may have a significant influence on the algorithm performance. Our computational results highlight the capabilities and the limitations of this approach.

Pros of IP column generation

The IP column generation approach seems to be well suited when most of the combinatorial difficulty of the set-partitioning-like optimization problem on hand is in the subproblem (which implies that the master LP bound is tight) but still the subproblem remains tractable.

We point out that we have obtained our results for the network design and clustering problems using very few application specific devices. Apart from the subproblem heuristics, the IP column generation algorithm we have implemented is a general purpose shell.

This approach is portable to other applications. To adapt the IP column generation algorithm to other set-partitioning-like optimization problems, one only needs to deal with the subproblem for which one might find existing results in the literature or for which one might easily develop a heuristic. By embedding the subproblem solution technique in the IP column generation algorithm, one often obtains a good (optimal) primal solution and a bound to attest of its quality.

Decomposing the optimization problem leads to a natural disaggregation and yields subproblems that are interesting on their own right. Then, the success of this approach depends essentially on one's ability to solve the subproblem. Developing an efficient solution method for the subproblem is hopefully much easier than tackling the global problem directly.

Cons of IP column generation

The method breaks down when the subproblem is intractable (i.e. if one cannot solve the subproblem within reasonable time). In many applications, the subproblem is difficult and cannot be solved with standard techniques. It is then necessary to develop a special purpose algorithm to solve the subproblem. For instance, in our experiment with the MILS application, we could not solve the subproblem without using cuts. Also, when the size of the problem data gets larger, the subproblem solution might require too much time as we observed in our computations for applications ND and CLUST.

Another drawback of the IP column generation algorithm arises when the solution of the master becomes intractable: For the network design problems with high ring capacity, we have encountered difficulties in finding an optimum master IP solution because there are many combinations of columns yielding the same objective value. We observed that although the master LP bounds are very tight, the branch-and-bound tree grows large because branching results in many alternative solutions of the same cost. For the equipartitioning clustering problems (*mesh*), the master LP itself is hard to solve due to the degeneracy.

Further Remarks

There is more work to be done to study the robustness of this methodology across applications and to see how this approach compares with others. In particular, we would like to test the IP column generation algorithm on applications for which the root master LP bound is not as close to the optimal IP solution. Also, it would be interesting to compare the branch-and-cut and the IP column generation approaches on a more complicated versions of the multi-item single-machine lot-sizing problem with set-up times.

The enhancements of branch-and-bound commonly used in integer programming

have not been fully exploited here. Most standard techniques used for solving integer programs have their analogue in an IP column generation context. For instance, let us mention variable fixing by reduced cost. If one applies it to the master variables λ, one encounters the same problem inherent to branching on λ (i.e. it is problematic to enforce $\lambda = 0$). Instead, one could apply the reduced cost fixing concept to subsets of master rows such as those we have used for branching [1]. Other standard techniques such as constructing a master integer solution by rounding of the LP solution can probably be adapted to the column generation context.

Due to the correspondence between adding columns in the primal and adding cuts in the dual master formulation, one can perhaps also draw on techniques used in branch-and-cut to develop IP column generation further.

Among the questions left unanswered, it seems interesting to find out if we can reduce the tailing off effect by using dual solutions that are central points of the optimal face (i.e. solutions obtained with an interior point method) rather than extreme point solutions.

We have seen that a good a priori bound on the optimum IP solution significantly helps in reducing the computational effort. Consequently, it sounds reasonable to spend more time in trying to find good integer master solutions early in the branch-and-bound procedure and to develop heuristics specific to the application for the master IP.

We have tested the use of a cutting plane algorithm for the subproblem. We have observed that, for applications ND and CLUST, the subproblem solution is quicker without these extra rows in the formulation. This raises the question if a full blown branch-and-cut algorithm for the subproblem would improve the overall performance of the algorithm, or if it is better to tackle the subproblem with a meta-heuristic that can handle the subproblem modifications due to branching.

Note that the branching scheme we have proposed can be viewed as a general procedure to add constraints to the master and modify the subproblem accordingly. In the same spirit, one can strengthen the master LP formulation to reduce the gap

[1] For instance, assume the current master LP solution at node u is $Z^u_{LP}(\tilde{Q})$. Solving the subproblem with additional constraint $x_i = x_j$ gives the best reduced cost of all patterns where i and j are together, $\overline{c}^{ij}_q$. Then, if $Z^u_{LP}(\tilde{Q}) + \overline{c}^{ij}_q > Z^{INC}$, one can restrict the search for an optimal solution to the subset of patterns such that $a_{iq} \neq a_{jq}$.

with the IP master by adding global cuts such as those proposed by Ferreira et. al. (1994) [17] for the clustering problem. However, it is important to make sure that the subproblem remains tractable after the modifications yielded by the master cuts. So, a rich topic for further research is the combination of IP column generation with branch-and-cut.

References

[1] E.H. Aghezzaf, T.L. Magnanti and L.A. Wolsey (1992). Optimizing Constrained Subtrees of Trees, *CORE Discussion Paper*, 9250, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

[2] E. Balas, S. Ceria, and G. Cornuéjols (1993). A lift-and-project cutting plane algorithm for mixed 0-1 program, *Mathematical Programming*, 58, 295-324.

[3] C. Barnhart, E.L. Johnson, and R. Anbil (1991). A Column Generation Technique for the Long-haul Crew Assignment Problem, Computational Optimization Center COC-91-01, Georgia Institute of Technology, Atlanta, Georgia.

[4] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh and P.H. Vance (1994). Branch-and-Price: Column Generation for Solving Huge Integer Programs, Computational Optimization Center COC-94-03, Georgia Institute of Technology, Atlanta, Georgia.

[5] C. Barnhart, E.L. Johnson, and G. Sigismondi (1991). An Alternative Formulation and Solution Strategy for Multi-Commodity Network Flow Problems, Computational Optimization Center COC-91-02, Georgia Institute of Technology, Atlanta, Georgia.

[6] S. Ceria (1984). Private Communication.

[7] M. Constantino (1993). A Cutting Plane Approach to Capacitate Lot-sizing with Start-up costs, *CORE Discussion Paper*, 9338, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

[8] V. Chvátal (1983). *Linear Programming*, W.H. Freeman and Co.

[9] *Using the CPLEX Linear Optimizer*, version 2.1. (1993). CPLEX Optimization, Inc., Suite 279, 930 Tahoe Blvd., Bldg. 802, Incline Village, NV 89451-9436. (702) 831-7744.

[10] Y. Crama and J.J. van de Klundert (1993). Approximation Algorithms for Integer Covering Problems via Greedy Column Generation, Research Memorandum RM 93-036, Faculty of Ecomomics, Limburg University, Maastricht.

[11] G.B. Dantzig and P. Wolfe (1960). Decomposition Principle for Linear Programs, *Operations Research* 8, 101-111.

[12] Z. Degraeve (1992). Scheduling joint product operations with proposal generation methods, Dissertation for the degree of Doctor in Philosophy, Graduate School of Business, University of Chicago, Chicago, Illinois.

[13] C.C. de Souza (1993). The Graph Equipartition Problem: Optimal Solutions, Extensions and Applications, Doctoral Thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

[14] C.C. de Souza, R. Keunings, L.A. Wolsey, and O. Zone (1994). A New Approach to Minimizing the Frontwidth in Finite Element Calculations, *Computer Methods in Applied Mechanics and Engineering* 111, 323-334.

[15] M. Desrochers, J. Desrosiers and M. Solomon (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, vol. 40, no. 2, pp. 342-354.

[16] J. Desrosiers, Y. Dumas, M.M. Solomon and F. Soumis (1994). Time Constrained Routing and Scheduling, Chapter 11 in *Handbooks in Operations Research and Management Sciences: Networks eds* M.E. Ball, T.L. Magnanti, C. Monma, and G.L. Nemhauser, North-Holland.

[17] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel and L. A. Wolsey (1994). The Node Capacitate Graph Partitioning Problems: Formulations and Valid Inequalities, *CORE Discussion Paper*, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

[18] M.L. Fisher (1981). The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Sciences*, vol. 27, no. 1.

[19] J.J. Forrest and D. Goldfarb (1992). Steepest-edge simplex algorithms for linear programming, *Mathematical Programming*, 57, 341-374.

[20] M.R. Garey and D.S. Johnson (1979). *Computers and Intractability: a guide to the theory of NP-Completeness*. W.H. Freeman and Company, San Francisco.

[21] A.M. Geoffrion (1974). Lagrangian Relaxation for Integer Programming, *Mathematical Programming Study* 2, 82-114.

[22] P.C. Gilmore and R.E. Gomory (1961). A linear Programming Approach to the Cutting Stock Problem, *Operations Research* 9, 849-859.

[23] P.C. Gilmore and R.E. Gomory (1963). A linear Programming Approach to the Cutting Stock Problem: Part II, *Operations Research* 11, 863-888.

[24] J.L. Goffin , A. Haurie, and J.P. Vial (1992). Decomposition and Nondifferentiable Optimization with the Projective Algorithm, *Management Science* 38, 284-302.

[25] P. Hansen, B. Jaumart and M. Poggi de Aragao (1992). Mixed Integer Column Generation and the Probabilistic Maximum Satisfiability Problem, *Proceedings of IPCO2*, Carnegie-Mellon University, Pittsburg, 165-180.

[26] M. Held and R.M. Karp (1970). The Traveling Salesman Problem and Minimum Spanning Tree, *Operations Research* 18, 1138-1162.

[27] M. Held and R.M. Karp (1971). The Traveling Salesman Problem and Minimum Spanning Tree: Part II, *Mathematical Programming* 1, 6-25.

[28] K. Hoffman and M. Padberg (1985). LP-based combinatorial problem solving, *Annals of Operations Research* 4, 145-194.

[29] E. Johnson, A. Mehrotra and G.L. Nemhauser (1993). Min-cut Clustering, *Mathematical Programming* 62, 133-152.

[30] M. Junger, A. Martin, G. Reinelt and R. Weismantel (1992). Quadratic 0/1 Optimization and a Decomposition approach for the Placement of Electronics Circuits, *Konrad-Zuse-Zentrum fur Informationstechnik Berlin Preprint* SC 92-10.

[31] L.S. Lasdon (1970). *Optimization Theory for Large Scale Systems*, Mac Millan, New-York.

[32] T.L. Magnanti, J.F. Shapiro, and M. H. Wagner (1976). Generalized Linear Programming solves the dual, *Management Science*, 22, pp. 1195-1203.

[33] T.L. Magnanti and L.A. Wolsey (1994). Optimal Trees, Chapter X in *Handbooks in Operations Research and Management Sciences: Networks eds* M.E. Ball, T.L. Magnanti, C. Monma, and G.L. Nemhauser, North-Holland.

[34] O. Marcotte (1985). The Cutting Stock Problem and Integer Rounding, *Mathematical Programming*, 33, 89-92.

[35] M. Minoux. Mathematical Programming: Theory and Algorithms. Wiley and Sons, NY.

[36] M. Minoux (1987). A Class of Combinatorial Optimization Problems with Polynomially Solvable Large Scale Set-Covering/Partitioning Relaxations, *RAIRO* 21, 105-136.

[37] C.C. Ribeiro, M. Minoux, and M.C, Penna (1989). An Optimal Column-Generation-with-Ranking Algorithm for Very Large Set Partitioning Problems in Traffic Assignment, *European Journal of Operational Research*, 41, 232-239.

[38] D.M. Ryan and B.A. Foster (1981). An Integer Programming Approach to Scheduling, A. Wren (ed.) *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, North-Holland, Amsterdam, 269-280.

[39] G.L. Nemhauser, M.W.P. Savelsbergh and G.C. Sigismondi (1994). MINTO, a Mixed INTeger Optimizer. *Operations Research Letters*, to appear.

[40] G.L. Nemhauser and L.A. Wolsey (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc.

[41] M.W.P. Savelsbergh (1993). A Branch-and-Price Algorithm for the Generalized Assignment Problem, Computational Optimization Center COC-93-03, Georgia Institute of Technology, Atlanta, Georgia.

[42] M.W.P. Savelsbergh and G.L. Nemhauser (1993). *Functional description of MINTO, a Mixed INTeger Optimizer*. Report COC-91-03A, , Georgia Institute of Technology, Atlanta, Georgia.

[43] J.F. Shapiro (1979). *Mathematical Programming, Structures and Algorithms*. John Wiley & Sons, Inc.

[44] P.H. Vance, C. Barnhart, E.L. Johnson, G.L. Nemhauser (1992). Solving Binary Cutting Stock Problem by Column Generation and Branch-and-Bound, Computational Optimization Center COC-92-09, Georgia Institute of Technology, Atlanta, Georgia.

[45] F. Vanderbeck (1993). A Decomposition Approach for Parallel Machine Assignment and Setup Minimization in Electronics Assembly, Dissertation for the degree of Master of Science in Operations Research, MIT Sloan School of Management, Boston, USA.

[46] F. Vanderbeck and L. A. Wolsey (1994). An Exact Algorithm for IP Column Generation, *CORE Discussion Paper 9419*, Université Catholique de Louvain, Louvain-la-Neuve, Belgium.

A Problem Formulations

In our implementation, we use a set covering formulation for the network design applications (ND and NDSA), and for the multi-item single-machine lot-sizing application (MILS). For the Clustering application (CLUST), we use a set packing formulation, assuming that the number of clusters is not prescribed. Below we present for each of these applications, the Master Problem, the Pricing Subproblem, the specific expression of the reduced cost of a pattern and the bound $LB^u(\pi, \mu, \nu)$ and the a priori bounds on the subproblem value $v^u(\pi, \mu, \nu)$.

The linear relaxation of the master is obtained by replacing the integrality constraints with non-negativity constraints. We do not define variable upper bounds in the LP relaxation as it perturbs the shadow prices and raise feasibility questions. The dual formulation of the master LP is explicitly given to show what drives the dual variables. A new column for the primal master is a new constraint in the dual master formulation.

Note that, since in the master problem we have replace the standard convexity constraints by an aggregated cardinality constraint (28), we may exclude the null vector from the set of feasible pattern. Due to the shadow price of constraint (29), the null pattern might be generated by the subproblem as the best reduced cost pattern leading to an unproductive iteration of the column generation algorithm. So we add an extra constraint in the subproblem that excludes the null solution. Also, we use additional constraints to strengthen the formulation of the subproblem.

A.1 The Network Design application (ND)

Master Problem

$$
\begin{array}{lrcll}
 & Z^u(Q) & = & \min & \displaystyle\sum_{q \in Q} c_q \lambda_q & \\
[M^u]^{ND} & & & \text{s.t.} & & (63) \\
 & \displaystyle\sum_{q \in Q} a_{e\,q} \lambda_q & \geq & 1 & \forall e \in E & \\
 & \displaystyle\sum_{q \in Q} \lambda_q & \leq & K^0 & & \\
 & \displaystyle\sum_{q \in Q} \lambda_q & \geq & L^0 & & \\
 & \lambda_q & \in & \{0,1\} & \forall q \in Q &
\end{array}
$$

with $\lceil \frac{\sum_{e \in E} d_e}{C} \rceil \leq L \leq L^0 \leq K^0 \leq K \leq \lfloor \frac{Z^{INC}}{2} \rfloor \leq |E|$. Indeed, since in any feasible solution each pattern contains at least 2 nodes, the cost (z^{INC}) of the best integer solution divided by 2 provides an upper bound on the maximum number of rings used in an optimal solution.

Dual formulation of the restricted master LP

$$
\begin{array}{lrll}
 & Z^u_{LP}(\tilde{Q}) \;=\; \max & \displaystyle\sum_{e \in E} \pi_e + \mu_0 K^0 + \nu_0 L^0 & \\
[M^u_{LP}]^{ND} & \text{s.t.} & & (64) \\
 & \displaystyle\sum_{e \in E} \pi_e a_{eq} + \mu_0 + \nu_0 \;\leq\; c_q & & \forall q \in \tilde{Q} \\
 & \pi_e \;\geq\; 0 & & \forall e \in E \\
 & \mu_0 \;\leq\; 0 & & \\
 & \nu_0 \;\geq\; 0 & &
\end{array}
$$

Column reduced cost

$$\overline{c_q} = c_q - \sum_{e \in E} \pi_e a_{eq} - \mu_0 - \nu_0$$

Pricing SubProblem

$$
\begin{array}{lrll}
 & v^u(\pi, \mu, \nu) = \min & \displaystyle\sum_{i \in V} y_i - \sum_{e \in E} \pi_e x_e - \mu_0 - \nu_0 & \\
[SP^u]^{ND} & \text{s.t.} & & (65) \\
 & y_i \;\geq\; x_e & & \forall e,\, i, \text{ with } e \in \delta(i) \\
 & y_i \;\leq\; \sum_{e \in \delta(i)} x_e & & \forall i, \\
 & \sum_{i \in V} y_i \;\geq\; 2 & & \\
 & \sum_{e \in E} d_e x_e \;\leq\; C & & \\
 & \sum_{e \in \delta(i)} d_e x_e \;\leq\; C y_i & & \forall i, \\
 & x_e \;=\; x_f & & \forall (e,f) \in G, \\
 & x_e + x_f \;\leq\; 1 & & \forall (e,f) \in H, \\
 & x_e,\; y_i \;\in\; \{0,1\} & & \forall\, e,\, i
\end{array}
$$

where $\delta(i)$ is the set of edges incident to node i and G (resp. H) is the set of pairs of edges that have to be assigned to the same ring (resp. different rings) according to the branching constraints.

Lower Bounds

$$\begin{aligned} LB^u(\pi,\mu,\nu) &= Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0 \\ &\quad + \min\{K^0(v^u(\pi,\mu,\nu)+\mu_0+\nu_0), L^0(v^u(\pi,\mu,\nu)+\mu_0+\nu_0)\} \end{aligned}$$

A priori bound on the subproblem value

Let $\alpha_1 = \lceil Z^u_{LP}(\tilde{Q})\rceil - 1 - TOL - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$ and $\alpha_2 = Z^{INC} - 1 - TOL - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$ where TOL is the tolerance in the approximation algorithm ($TOL = 0$ in the exact algorithm).

$$\begin{aligned} v^u(\pi,\mu,\nu) &< 0 \\ v^u(\pi,\mu,\nu) &\leq \max\{\tfrac{\alpha_1}{K^0}, \tfrac{\alpha_1}{L^0}\} - \mu^0 - \nu^0 \\ v^u(\pi,\mu,\nu) &\leq \max\{\tfrac{\alpha_2}{K^0}, \tfrac{\alpha_2}{L^0}\} - \mu^0 - \nu^0 \end{aligned}$$

A.2 Network Design application with Split Assignment (NDSA)

Master Problem

$$[M^u]^{NDSA} \qquad \begin{aligned} Z^u(Q) = \min \quad & \sum_{q\in Q} c_q \lambda_q & & \\ \text{s.t.} \quad & & & \\ & \sum_{q\in Q} a_{e\,q}\lambda_q \geq b_e & \forall e \in E' & \\ & \sum_{q\in Q: a_q \geq a_g} \lambda_q \leq K^g & \forall g \in G^u & \\ & \sum_{q\in Q: a_q \geq a_h} \lambda_q \geq L^h & \forall h \in H^u & \\ & \lambda_q \in \{0,1,2\} & \forall q \in Q & \end{aligned} \tag{66}$$

with $\lceil \frac{\sum_{e\in E'} d'_e}{C} \rceil \leq L \leq L^0 \leq K^0 \leq K \leq \lfloor \frac{Z^{INC}}{2} \rfloor \leq |E'|$, where E' and d'_e represent respectively the edge set and the associated demands after the demand splitting transformation.

Dual formulation of the restricted master LP

$$[M^u_{LP}]^{NDSA} \qquad \begin{aligned} Z^u_{LP}(\tilde{Q}) = \max \quad & \sum_{e\in E'} \pi_e + \sum_{g\in G^u} \mu_g K^g + \sum_{h\in H^u} \nu_h L^h & & \\ \text{s.t.} \quad & & & \\ \sum_{e\in E'} \pi_e a_{e\,q} + \sum_{g\in G^u: a_q \geq a_g} \mu_g + \sum_{h\in H^u: a_q \geq a_h} & \nu_h \leq c_q & \forall q \in \tilde{Q} & \\ & \pi_e \geq 0 & \forall e \in E' & \\ & \mu_g \leq 0 & \forall g \in G^u & \\ & \nu_h \geq 0 & \forall h \in H^u & \end{aligned} \tag{67}$$

Column reduced cost

$$\overline{c_q} = c_q - \sum_{e \in E'} \pi_e a_{e\,q} - \sum_{g \in G^u : a_q \geq a_g} \mu_g - \sum_{h \in H^u : a_q \geq a_h} \nu_h$$

Pricing SubProblem

$$\begin{aligned}
v^u(\pi,\mu,\nu) = \min \quad & \sum_{i \in V} y_i - \sum_{e \in E'} \pi_e x_e - \mu_0 - \nu_0 - \sum_{g \in G^u_*} \mu_g w^g - \sum_{h \in H^u_*} \mu_h w^h \\
[SP^u]^{NDSA} \qquad \text{s.t.} \quad & \qquad (68) \\
y_i \;&\geq\; x_e && \forall e,\ i, \text{ with } e \in \delta(i) \\
y_i \;&\leq\; \textstyle\sum_{e \in \delta(i)} x_e && \forall i, \\
\textstyle\sum_{i \in V} y_i \;&\geq\; 2 \\
\textstyle\sum_{e \in E'} d_e x_e \;&\leq\; C \\
\textstyle\sum_{e \in \delta(i)} d_e x_e \;&\leq\; C y_i && \forall i, \\
x_e \;&\leq\; x_f && \forall (e,f) \in O, \\
x_e + x_f \;&\leq\; 1 && \forall (e,f) \in P, \\
x_e + x_f + x_g \;&\leq\; 2 && \forall (e,f,g) \in R, \\
w^j \;&\leq\; x_e && \forall e \in T^j \subseteq I, \forall j \in G^u_* \cup H^u_*, \\
w^j \;&\geq\; 1 - \textstyle\sum_{e \in T^j} (1 - x_e) && \forall j \in G^u_* \cup H^u_*, \\
x_e,\ y_i \;&\in\; \{0,1\} && \forall\, e \in E',\ i \in V
\end{aligned}$$

where O, P and R define set of constraints resulting from special cases of branching constraints, and G^u_* and H^u_* result from general cases of branching. Moreover P includes the initial set of edge disjunctive constraints.

Lower Bounds

$$\begin{aligned}
LB^u(\pi,\mu,\nu) \;=\; & Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0 \\
& + \min\{K^0(v^u(\pi,\mu,\nu) + \mu_0 + \nu_0),\, L^0(v^u(\pi,\mu,\nu) + \mu_0 + \nu_0)\}
\end{aligned}$$

A priori bound on the subproblem value

Let $\alpha_1 = \lceil Z^u_{LP}(\tilde{Q}) \rceil - 1 - TOL - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$ and $\alpha_2 = Z^{INC} - 1 - TOL - Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0$ where TOL is the tolerance in the approximation algorithm ($TOL = 0$ in the exact algorithm).

$$\begin{aligned}
v^u(\pi,\mu,\nu) \;&<\; 0 \\
v^u(\pi,\mu,\nu) \;&\leq\; \max\{\tfrac{\alpha_1}{K^0}, \tfrac{\alpha_1}{L^0}\} - \mu^0 - \nu^0 \\
v^u(\pi,\mu,\nu) \;&\leq\; \max\{\tfrac{\alpha_2}{K^0}, \tfrac{\alpha_2}{L^0}\} - \mu^0 - \nu^0
\end{aligned}$$

A.3 The Clustering application (CLUST)

Master Problem

$$
\begin{array}{lrcll}
 & Z^u(Q) & = & \max & \displaystyle\sum_{q\in Q} c_q\,\lambda_q & \\
[M^u]^{CLUST} & & & \text{s.t.} & & (69)\\
 & \displaystyle\sum_{q\in Q} a_{i\,q}\,\lambda_q & \leq & 1 & \forall i \in V & \\
 & \displaystyle\sum_{q\in Q} \lambda_q & \leq & K^0 & & \\
 & \displaystyle\sum_{q\in Q} \lambda_q & \geq & L^0 & & \\
 & \lambda_q & \in & \{0,1\} & \forall q \in Q &
\end{array}
$$

with $\lceil \frac{\sum_{i\in V} d_i}{C} \rceil \leq L \leq L^0 \leq K^0 \leq K \leq |V|$.

Dual formulation of the restricted master LP

$$
\begin{array}{lrcll}
 & Z^u_{LP}(\tilde{Q}) & = & \min & \displaystyle\sum_{i\in V} \pi_i \,-\, \mu_0 K^0 \,-\, \nu_0 L^0 & \\
[M^u_{LP}]^{CLUST} & & & \text{s.t.} & & (70)\\
 & \displaystyle\sum_{i\in V} \pi_i a_{i\,q} \,-\, \mu_0 \,-\, \nu_0 & \geq & c_q & \forall q \in \tilde{Q} & \\
 & \pi_i & \geq & 0 & \forall i \in V & \\
 & \mu_0 & \leq & 0 & & \\
 & \nu_0 & \geq & 0 & &
\end{array}
$$

Column reduced cost

$$\overline{c_q} = c_q \,-\, \sum_{i\in V} \pi_i a_{i\,q} \,+\, \mu_0 \,+\, \nu_0$$

Pricing SubProblem

$$
\begin{array}{lrcll}
 & v^u(\pi,\mu,\nu) = \max & & & \displaystyle\sum_{e\in E} c_e y_e - \sum_{i\in V} \pi_i x_i + \mu_0 + \nu_0 & \\
[SP^u]^{CLUST} & & & \text{s.t.} & & (71)\\
 & y_e & \leq & x_i & \forall e,\ i, \text{ with } e \in \delta(i) & \\
 & y_e & \geq & x_i + x_j - 1 & \forall e, i, j : e \in \delta(i) \cap \delta(j), & \\
 & \textstyle\sum_{i\in V} x_i & \geq & 1 & & \\
 & \textstyle\sum_{i\in V} d_i x_i & \leq & C & & \\
 & \textstyle\sum_{e\in\delta(i)\cap\delta(j)} d_j y_e & \leq & (C - d_i)x_i & \forall i, &
\end{array}
$$

$$
\begin{aligned}
x_i &= x_j && \forall (i,j) \in G, \\
x_i + x_j &\leq 1 && \forall (i,j) \in H, \\
x_i &\in \{0,1\} && \forall i \in V \\
y_e &\geq 0 && \forall\, e \in E
\end{aligned}
$$

where $\delta(i)$ is the set of edges incident to node i and G (resp. H) is the set of pairs of nodes that have to be assigned to the same cluster (resp. different clusters) according to the branching constraints.

Upper Bounds

$$
\begin{aligned}
UB^u(\pi,\mu,\nu) &= Z^u_{LP}(\tilde{Q}) + \mu_0 K^0 + \nu_0 L^0 \\
&\quad + \max\{K^0(v^u(\pi,\mu,\nu) - \mu_0 - \nu_0), L^0(v^u(\pi,\mu,\nu) - \mu_0 - \nu_0)\}
\end{aligned}
$$

A priori bound on the subproblem value
Let $\alpha_1 = \lfloor Z^u_{LP}(\tilde{Q}) \rfloor + 1 + TOL - Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0$ and $\alpha_2 = Z^{INC} + 1 + TOL - Z^u_{LP}(\tilde{Q}) - \mu_0 K^0 - \nu_0 L^0$ where TOL is the tolerance in the approximation algorithm ($TOL = 0$ in the exact algorithm).

$$
\begin{aligned}
v^u(\pi,\mu,\nu) &> 0 \\
v^u(\pi,\mu,\nu) &\geq \min\{\tfrac{\alpha_1}{K^0}, \tfrac{\alpha_1}{L^0}\} + \mu^0 + \nu^0 \\
v^u(\pi,\mu,\nu) &\geq \min\{\tfrac{\alpha_2}{K^0}, \tfrac{\alpha_2}{L^0}\} + \mu^0 + \nu^0
\end{aligned}
$$

A.4 The single-machine Multi-Item Lot-Sizing application (MILS)

Master Problem

$$
[M^u]^{MILS} \qquad
\begin{aligned}
Z^u(Q) = \min \quad & \sum_{q \in Q} c_q \lambda_q \\
\text{s.t.} \quad & \\
\sum_{q \in Q} a_{t\,q} \lambda_q &\leq 1 && \forall t = 1, \ldots, T \\
\sum_{q \in Q} a_{(T+i)\,q} \lambda_q &\geq 1 && \forall i = 1, \ldots, I \\
\lambda_q &\in \{0,1\} && \forall q \in Q
\end{aligned}
\tag{72}
$$

Dual formulation of the restricted master LP

$$
Z^u_{LP}(\tilde{Q}) = \max \quad \sum_{i=1}^{I} \pi_i - \sum_{t=1}^{T} \pi_t
$$

$$
\begin{array}{lll}
[M^u_{LP}]^{MILS} & \text{s.t.} & \\
\sum_{i=1}^{I} \pi_i a_{i\,q} - \sum_{t=1}^{T} \pi_t a_{t\,q} & \leq & c_q \qquad \forall q \in \tilde{Q} \\
\pi_i & \geq & 0 \qquad \forall i = 1, \ldots, I \\
\pi_t & \geq & 0 \qquad \forall t = 1, \ldots, T
\end{array} \tag{73}
$$

Column reduced cost

$$
\overline{c_q} = c_q + \sum_{t=1}^{T} \pi_t a_{t\,q} - \sum_{i=1}^{I} \pi_i a_{i\,q}
$$

Pricing SubProblem

$$
\begin{array}{lll}
v^u_i(\pi) = \min & \sum_{t=1}^{T}((c^i_t + \pi_t)\, x_t + f^i_t\, y_t + p^i_t\, z_t + h^i_t\, s_t) - \pi_i & \\
[SP^u_i]^{MILS} \quad \text{s.t.} & & \\
z_t + s_t = d^i_t + s_{t+1} & & \forall t \\
z_t \leq U^i_t\, x_t & & \forall\, t \\
y_t \geq x_t - x_{t-1} & & \forall\, t \\
s_t \geq d^i_t\,(1 - x_t) & & \forall\, t \\
x_t \leq 0 & & \forall\, t \in G^u_i \\
x_t \geq 1 & & \forall\, t \in H^u_i \\
x_t,\ y_t \in \{0, 1\} & & \forall\, t
\end{array} \tag{74}
$$

where G^u_i (resp. H^u_i) represent the set of period for which the machine is dedicated to (resp. unavailable for) item i as a result of the branching constraints.

Lower Bounds

$$
\begin{array}{lll}
LB^u(\pi, \mu, \nu) & = & Z^u_{LP}(\tilde{Q}) + I \min_i v^u_i(\pi) \\
\text{or} \quad LB^u(\pi, \mu, \nu) & = & Z^u_{LP}(\tilde{Q}) + \sum_{i=1}^{I} v^u_i(\pi)
\end{array}
$$

where the first expression results from a strict applications of the general lower bound we have defined, while the second is specific to the case where the subsystem have different characteristics and is a little stronger.

A priori bound on the subproblem value

Let $\alpha_1 = \lceil Z^u_{LP}(\tilde{Q}) \rceil - 1 - TOL - Z^u_{LP}(\tilde{Q})$ and $\alpha_2 = Z^{INC} - 1 - TOL - Z^u_{LP}(\tilde{Q})$ where TOL is the tolerance in the approximation algorithm ($TOL = 0$ in the exact algorithm).

$$\begin{aligned} v_i^u(\pi) &< 0 \\ v_i^u(\pi) &\leq \frac{\alpha_1}{I} \\ v_i^u(\pi) &\leq \frac{\alpha_2}{I} \end{aligned}$$

As long as there is an item i for which the subproblem value satisfies these bounds, column generation should continue. If for all item i one of these 3 bounds is violated, column generation terminates.

B Program Description

IPCG (Integer Programming Column Generation) is a computer program coded in C. It solves an integer program, called the master, of the set partitioning type, having many columns. Note that the code treats set covering and set packing problems as well. The master problem is solved to optimality by branch-and-bound with linear programming based relaxation.

The columns of the master are only known implicitly. They are solutions of an application specific mixed-integer subproblem. Consequently, a column generation procedure is used to solve the linear relaxation of the master problem at each node of the branch-and-bound tree. The branching scheme is specifically designed to be compatible with column generation. A judicious use of bounds attempt to reduce the column generation tailing-off effect.

This program is build on top of the CPLEX callable library, which handles the master LP optimization and the subproblem LP and IP optimization. The code is at present customized for 3 applications: i) a network design problem, referred to as ND, which is a set covering minimization problem; ii) a graph partitioning problem, referred to as CLUST, which is a set packing maximization problem; iii) a single-machine multi-item lot-sizing problem, referred to as MILS, which is a minimization set packing problem with different types of columns and subproblems.

The instructions on how to use the program, how to setup the inputs and collect the outputs, how to modify its parameters and options, are given in a companion file named readme. The code is composed of many subroutines, each of which is contained in a separate file whose name is the same as the subroutine name preceded by "CG" and ended by ".c". Header files (with extension ".h") contain the global variable declarations, the subroutine prototypes and other program parameters.

Next we give a chart picturing where each subroutine is called. We give each subroutine name in the order in which they appear in the program. Below any given subroutine, we place a tab and list all the subroutines that are called by it (this is done recursively). At the end of that chart we give the list of utility subroutines which are called throughout the program.

```
************************
THE PROGRAM
************************

main()
    readOptions();
    initializations();
        printOptions();
        readdata();
            readNDdata();
            readCLUSdata();
            readMILSdata();
        initVar();
            initNDVar();
                createNewNode();
                    addNodeToActiveList();
            initCLUSVar();
                createNewNode();
                    addNodeToActiveList();
                initCLUSseparation();
            initMILSVar();
                createNewNode();
                    addNodeToActiveList();
                initMILSseparation();
        generateMaster();
        initCplexMastVar();
        readAndLoadMast();
        generateNDSubProblem();
        generateCLUSTSubProblem();
        generateMILSSubProblem();
        initCplexSuPbVar();
        readAndLoadSuPb();
        readInitialSolution();
    selectNode();
        readCustomization();
            defineGroups();
                computeGroupCharacterictics();
        detectInconsistency();
        addGroupCol();
            addCandidatColToRec();
            recordNewCols();
            addVarsInMaster();
        initSetofCol();
            addCandidatColToRec();
            recordNewCols();
```

```
            addVarsInMaster();
        customizeMaster();
            delVarsInMaster();
            addVarsInMaster();
        customizeSubproblem();
solveMaster();
        solveLP();
        getDualValues();
        checkMasterSol()
        compareWithIncumbent();
terminateNode();
findNewCols();
        solveSubproblem();
            computeSpLowerBd();
            solveSpHeuristically();
                solveNDSpConstrHeur();
                solveCLUSTSpConstrHeur();
            updateSubproblem();
            computeSpUpperBd();
                solveLP();
                checkSPfeasibility();
                spCuttingPlaneAlg();
                    checkSpCutPool();
                    solveLP();
                    checkSPfeasibility();
                    cutoffSPfractSol();
                        knapNDSpcuts();
                        knapCLUSTSpcuts();
                        knapTreeCLUSTSpcuts();
                        milsSpCuts();
                            cssep1();
                            sepfam3();
                            cssep();
                        addSpCutToPool();
                    retrieveSpSol();
                addCandidatColToRec();
            solveSPoptimally();
                solveMIP();
                checkSPfeasibility();
                retrieveSpSol();
                addCandidatColToRec();
            eraseAllSpCuts();
        recordNewCols();
        addVarsInMaster();
        updateGap();
```

```
phase1uncompleted()
updateBBtree();
        deleteGroupCustomization();
        uncustomizeSubproblem();
        eraseAllSpCuts();
        uncustomizeMaster();
        updateGap();
        divideNode();
                chooseProductsForBranch();
                createNewNode();
                        addNodeToActiveList();
        solveRestMastIPbyBB();
                solveMIP();
                compareWithIncumbent();
        constrHeurPrimalSol();
                compareWithIncumbent();
        fathomByBDnodeList();
printOut();
        printSolution();
        printTreeNodes();
        printOptions();
        printTree();
        printRes();
        printStatistics();
        printTimes();
closing();
        eraseCplexMastVar();
        eraseCplexSuPbVar();


**********************
SUBROUTINE LIBRARY
**********************

        C standard funtion
        CPLEX library
        declar_dyn Library
        checkSpace Library

        and

        convertDtoL();
        readword();
        tempo();
```

terminateEarly();
knapsackcutfor_();
CGdijkstra();

****
HEADER FILES
****

CGdeclar_dyn.h
CGglobForCplexMastDeclar.h
CGglobForCplexMastExt.h
CGglobForCplexSuPbDeclar.h
CGglobForCplexSuPbExt.h
CGglobalDeclar.h
CGglobalExt.h
CGincl_standard.h
CGparamForConstant.h
CGparamForSwitch.h
CGsubroutPrototype.h
CGtyp_def.h

****
MAKE FILE
****

MakeCG

C Data Sample

ND7c60

number of nodes: 7

ring capacity: 60

demand matrix:

8	25	6	13	5	7
	11	4	5	3	2
		13	38	11	18
			39	31	17
				32	29
					19

ND8c60

number of nodes: 8

ring capacity: 60

demand matrix:

10	44	2	9	29	28	6
	15	10	15	56	14	11
		10	20	21	25	3
			9	24	25	14
				52	16	10
					18	46
						35

ND9c60

number of nodes: 9

ring capacity: 60

demand matrix:

10	8	10	10	68	20	13	34
	9	10	11	48	17	19	38
		7	8	51	13	14	35
			9	50	16	13	44
				45	19	14	26
					90	105	117
						27	48
							65