UNIVERSIDAD DE LA LAGUNA

«Cycle location problems»

Autor: Inmaculada Rodríguez Martín
Director: Dr. D. José Ándres Moreno Pérez

Departamento de Estadística,
Investigación Operativa y Computación

MARTINE LABBÉ, CATEDRÁTICA DEL "INSTITUT DE STATISTIQUE ET DE RECHERCHE OPÉRATIONNELLE" DE LA "UNIVERSITÉ LIBRE DE BRUXELES" (BRUSELAS, BÉLGICA)

JOSÉ ANDRÉS MORENO PÉREZ, CATEDRÁTICO DEL ÁREA DE CIENCIAS DE LA COMPUTACIÓN E INTELIGENCIA ARTIFICIAL DE LA UNIVERSIDAD DE LA LAGUNA

CERTIFICAMOS:

Que la presente memoria, titulada "Cycle Location Problems", ha sido realizada bajo nuestra dirección por la licenciada Dña. Inmaculada Rodríguez Martín, y constituye su Tesis para optar al grado de Doctor por la Universidad de La Laguna.

Y para que conste, en cumplimiento de la legislación vigente y a los efectos que haya lugar, firmamos la presente en La Laguna, a nueve de Marzo de dos mil.

Fdo.: Martine Labbé

Fdo.: José A. Moreno Pérez

A mis padres y hermano

This memory has been done under the direction of professors José A. Moreno Pérez and Martine Labbé, to whom I want to express my most sincere gratitude. I thank professor José A. Moreno Pérez for having introduced me in the field of scientific research, and having supported me and guided my steps so wisely all these years. My debt with professor Martine Labbé is big, and I feel particularly grateful for the time she has dedicated to this thesis, for her patience, for the encouragement she has always given me, and for her constant help and availability.

I also thank professors Juan José Salazar González and Gilbert Laporte for their many contributions to this work and good advices.

Finally, I cannot forget in this section about all my colleagues and friends of the Departamento de Estadística, Investigación Operativa y Computación, and those of the Service de Mathématiques de la Gestion.

To all, thank you.

Contents

Contents **i**

Introduction **v**

1 General concepts **1**

1.1 Graph theory 1

1.2 Computational complexity 3

1.3 Polyhedral theory 5

1.4 Polyhedral combinatorics 8

1.5 Multicriteria optimization 11

2 Path, Tree and Cycle location **13**

2.1 Introduction 13

2.2 Notation and formulations 14

2.3 Paths with fixed end points 20

2.4 Paths without fixed end points 23

2.5 Trees 24

2.6 Cycles with an origin 28

2.7 Cycles without a fixed origin 30

3 Two Median Cycle Problems **37**

3.1 Introduction and motivation 37
3.2 Applications and related problems 38
3.3 Model MCP1 40
3.3.1 Computational complexity 40
3.3.2 Mathematical formulation 41
3.3.3 Strengthening the LP-relaxation 43
3.4 Model for MCP2 50
3.4.1 Computational complexity 50
3.4.2 Mathematical formulation 51
3.4.3 Strengthening the LP-relaxation 52

4 Polyhedral study of MCP1 **59**
4.1 Dimension and trivial facets 61
4.2 Other facets 66
4.3 Small MCP1 polytopes 72
4.3.1 MCP1 polytope when $|V| = 3$ 73
4.3.2 MCP1 polytope when $|V| = 4$ 73
4.3.3 MCP1 polytope when $|V| = 5$ 74

5 Branch-and-Cut algorithms **77**
5.1 Separation procedures 77
5.1.1 Generalized subtour elimination constraints 77
5.1.2 2-matching inequalities 78
5.1.3 Cover-connectivity inequalities 80
5.2 Branch-and-Cut implementation 88
5.2.1 Preprocessing 88
5.2.2 Initialization 88
5.2.3 Feasibility check 90

5.2.4 Cutting plane phase 90
5.2.5 Branching strategy 92
5.2.6 Heuristic procedures 92

6 Variable Neighbourhood Tabu Search heuristic 97
6.1 The VNTS Heuristic 97
6.2 General aspects of the application of VNTS to Location-Allocation problems 101
6.2.1 Coding of the solutions 101
6.2.2 Evaluation of the solutions 101
6.2.3 The moves 102
6.2.4 Initial solution 102
6.3 The VNTS for MCP 103
6.3.1 Coding of the routes 103
6.3.2 The moves 104
6.3.3 The initial solution 107
6.3.4 Tabu list 108
6.3.5 Shake procedure 109
6.3.6 Local search 109
6.3.7 The strategy for the sizes 110
6.3.8 Stopping criterion 110

7 Computational results 111
7.1 Branch-and-Cut results 112
7.2 Heuristic results 122

A ABACUS implementation 129

Bibliography 140

Introduction

Traditional network location theory is concerned with the optimal location of facilities which can be considered as single points (emergency medical service stations, switching centers in communication networks, bus stops, mail boxes, etc.) However, in many real problems the facility to be located is too large to be modeled as a point. Examples of such problems include the location of pipelines and high speed train lines, the design of emergency routes, newspaper delivery routes, subway lines, etc. . We will refer to this kind of facilities as *extensive facilities* or *structures*, and they may have the shape of a path, a tree, a cycle or a more general subgraph.

Applications of extensive facilities location models can be found in location analysis, routing, and network design. As an example, consider the problem of locating a path connecting two given vertices of a graph with the objectives of minimizing the total path length and minimizing the total travel distance between some given demand points and the path (Current et al. [15]). The necessary information for solving this problem includes the distances given by the lengths of the shortest paths from each demand point to the closest vertex on the facility. That is, once the path is located it is assumed that the clients take the shortest routes to reach the facility. So, this problem can be viewed as a location problem, a network design problem, or a routing problem.

Many of these problems have been formulated as single objective problems (Traveling Salesman Problem, Shortest Path Problem, Minimum Spanning Tree Problem, etc.), being the minimization of the routing cost or length the criterium most frequently used. However, in the last twenty years there has been a growing interest in multiobjective formulations, to make the mathematical models closer to reality. One possible multiobjective approach is to consider the trade-off between the facility routing cost and the accessibility

of demand vertices, two objectives that are generally in conflict.

The most usual ways of measuring accessibility are the *total travel distance* (or *sum of distances*) that clients must travel to reach the nearest vertex in the facility (an extension of the median criterion) and the *coverage* of the facility. In the context of network design, routing and location of extensive facilities, two types of coverage appear: *direct* and *indirect*. Under direct coverage a vertex is said to be covered only if it is in the facility. Under indirect coverage a vertex is said to be covered if it is within a covering distance r of any vertex in the facility.

Finally, in many recent papers, biobjective location problems of extensive facilities involving routing cost minimization and accessibility maximization (minimizing the sum of distances or maximizing coverage) have been formulated as integer programs.

This dissertation is devoted to the study of the Median Cycle Problem (MCP), consisting of locating a cycle in a graph, visiting a given vertex, taking into account routing cost and total distance criteria.

In the first chapter some necessary and basic concepts are given. In Chapter 2 we propose generic integer programming formulations for bicriteria location problems involving path, tree and cycle-shaped structures. We review the existing literature on this topic.

In Chapter 3 we present two versions of the MCP. In the first one, MCP1, the objective function to minimize is a linear combination of the routing cost (length) and accessibility cost (sum of distances) of the facility. In the second version, MCP2, the objective is to minimize the routing cost of the facility while imposing an upper bound of its accessibility cost. We show that both problems are $\mathcal{NP}$-hard in the strong sense. We give integer programing formulation and show how to reinforce their LP-relaxations using valid inequalities.

Chapter 4 is devoted to the polyhedral study of MCP1. We derive dimension and general facet defining results. A separate study of the MCP1 polytope is small dimensional spaces is done. The obtained results are embedded into the Branch-and-Cut algorithm described in detail in Chapter 5.

In the first part of Chapter 6 we describe a new metaheuristic technique that combines Variable Neighbourhood Search and Tabu Search. In the

second part, we describe the application of the resulting method to solve the MCP.

Chapter 7 shows the computational results obtained by the exact and heuristic algorithms described in the previous chapters, and makes a comparison of the proposed metaheuristic with other heuristic methods.

Chapter 1

General concepts

This chapter gathers some basic concepts from graph theory, computational complexity, polyhedral theory, polyhedral combinatorics and multicriteria optimization that will be used in the forthcoming chapters of this dissertation.

1.1 Graph theory

An *undirected graph* or *network* $G = (V, E)$ consists of a finite set of *vertices* V, which are usually denoted by $v_1, v_2, .., v_n$, and a set E of *edges*. The edges are pairs of non-ordered vertices. We will denote by n the number of vertices and by m the number of edges of the graph. We will consider only *simple* graphs, i.e., graphs containing at most one edge linking each pair of vertices.

If $G = (V, E)$ is a graph and $e = [v_i, v_j] \in E$, then we say that v_i is *adjacent* to v_j (and vice-versa), and that e is *incident* to v_i and v_j. The set of all edges incident to vertex v_i is denoted by $\delta(i)$. The *degree* of a vertex v_i is the number of edges incident to v_i, i.e., $|\delta(i)|$. A graph is *complete* if it contains all possible edges. Given a set $V' \subseteq V$, we denote by $E(V')$ the set of all edges with both end vertices in V', that is, $E(V') = \{e = [v_i, v_j] \in E : v_i, v_j \in V'\}$. The *cut* $\delta(V')$ is the set of edges that are incident to exactly one vertex in V', in other words, $\delta(V') = \{e = [v_i, v_j] \in E : v_i \in V'$ and $v_j \notin V'\}$. A graph is *bipartite* if there is a cut that contains all its edges.

A *subgraph* of a graph $G = (V, E)$ is another graph $G' = (V', E')$ such

that $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$ then G' is a *spanning subgraph*. Given $V' \subseteq V$, $G = (V, E(V'))$ is the *subgraph induced* by V'.

A *path* in G is a sequence of vertices $v_1, v_2, ..., v_k$ such that $[v_i, v_{i+1}] \in E$ for $i = 1, .., k-1$. A path is *simple* if it contains no repeated vertices. A graph is *connected* if there is a path linking every pair of vertices of V. The maximal connected subgraphs of a graph are called *connected components.*

We say that a graph is *non-separable* if it is connected and cannot be disconnected by removing a single vertex (a cutvertex). A *block* is a maximal non-separable subgraph. A *cactus* is a graph in which each block is a single edge or a cycle

The *distance* $d(v_i, v_j)$ between two vertices v_i, $v_j \in V$ is equal to the length of the shortest path joining them.

A *cycle* is a path whose first and last vertices coincide. A cycle is *simple* if, except for the first and last vertices, no vertex appears more than once. A cycle is called *Hamiltonian* if it is simple and it visits all the vertices of the graph.

An *acyclic* graph or *forest* is a graph that does not contain any cycle A *tree* is a connected forest. The following proposition gives a useful characterization of trees.

Proposition 1 *Let $G = (V, E)$ be a graph with n vertices. The following statements are equivalent:*

1. *G is a tree.*
2. *There is a unique path between each pair of vertices in G.*
3. *G contains n-1 edges and it is connected.*
4. *G contains n-1 edges and it is acyclic.*
5. *G is connected and acyclic.*

A *directed graph* or *diagraph* is a pair $G = (V, A)$ where V is a set of vertices and A is a set of ordered pairs of vertices called *arcs*. If $(v_i, v_j) \in A$, then it is said that v_i is the *tail* and v_j is the *head* of the arc.

For more information about graph theory concepts we refer the reader to the text books of Berge [6], Christofides [8], and Bondy and Murty [7].

1.2 Computational complexity

A *problem* is a general question to be answered, which can have several parameters or variables whose values are left open. A problem is defined by giving a description of all its parameters and by specifying what properties a solution is required to satisfy. If all parameters are set to certain values, we speak of an *instance* of the problem. Formally, we assume that we have an *encoding scheme* which allows us to represent each instance of the problem (input) and each solution (output) as finite strings ρ and σ, respectively, of symbols of a finite alphabet Σ. The *size* of an instance is the length of the input string ρ.

An *algorithm* is a general non-ambiguous step-by-step procedure for solving a problem. An algorithm is said to *solve* a problem Π if it always finds a solution when applied to an instance I of Π.

To have a measure of the efficiency of an algorithm, we associate with it a function $T : \mathbb{N} \rightarrow \mathbb{N}$ called *time complexity function* which gives, for each $n \in \mathbb{N}$, the maximum number of steps that the algorithm needs to solve a problem instance of that size.

We say that a function $f(n)$ is $O(g(n))$ when there exists a constant $c \geq 0$ and an integer value N such that $\mid f(n) \mid \leq c \mid g(n) \mid$ for all $n \geq N$. An algorithm is said to be *polynomial* if its time complexity function is $O(p(n))$ for some polynomial p, otherwise it is said to be *exponential*. A *pseudo-polynomial* algorithm is an exponential algorithm that becomes polynomial if a bound on the magnitude of the numbers defining the instances is imposed. A problem is said to be *polynomial* if there exists a polynomial time algorithm for solving it.

A *decision problem* is a problem whose question requires only a 'yes' or 'no' answer. The class $\mathcal{P}$ consists of all those decision problems for which a polynomial time algorithm exists. A decision problem is said to be $\mathcal{NP}$ (or belong to $\mathcal{NP}$) when it is possible to *verify* in polynomial time if a given input will produce a yes-answer, for all inputs that effectively produce it. Note that the definition of $\mathcal{NP}$ does not imply polynomial resolution, but just polynomial 'verificability'. Clearly $\mathcal{P} \subseteq \mathcal{NP}$. Whether $\mathcal{P} = \mathcal{NP}$ remains an open question.

A problem Π_1 can be *polynomially reduced* to another problem Π_2 if there

exists a polynomial time algorithm that transforms each instance of Π_1 into an instance of Π_2. A decision problem Π is *$\mathcal{NP}$-complete* if $\Pi \in \mathcal{NP}$ and any other problem in $\mathcal{NP}$ can be polynomially reduced to Π. The term *$\mathcal{NP}$-hard* is reserved for the problems having this same property, even though they do not belong to $\mathcal{NP}$ or are not even decision problems. The class $\mathcal{NPC}$ of $\mathcal{NP}$-complete problems can be considered as the class of the most difficult problems in $\mathcal{NP}$.

A problem is *$\mathcal{NP}$-complete in the strong sense* if it cannot be solved by a pseudo-polynomial algorithm unless $\mathcal{P} = \mathcal{NP}$. Figure 1.1 shows the relation among $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NPC}$ and $\mathcal{NPC}$ in the strong sense, assuming $\mathcal{P} \neq \mathcal{NP}$.

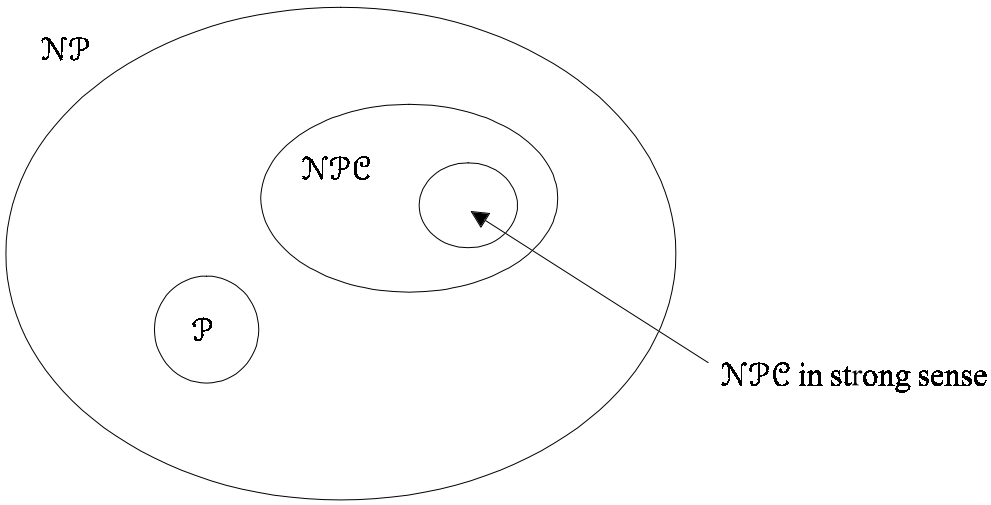

Figure 1.1: The world of $\mathcal{NP}$

One way of coping with $\mathcal{NP}$-complete problems is by using approximation algorithms. An algorithm A is an $(1+\varepsilon)$-*approximation,* $\varepsilon > 0$, algorithm for a problem Π if, given any instance I of Π, it finds a candidate solution $A(I)$ whose relation with the optimal solution $opt(I)$ is

$$\frac{opt(I)}{A(I)} \leq 1 + \varepsilon.$$

The theory of $\mathcal{NP}$-completeness is formulated, because of simplicity, for decision problems, while most interesting problems are optimization problems (in which a value must be minimized or maximized). However, any optimization problem can be easily reformulated as a decision problem by imposing a bound on the value to be optimized, and it can be addressed by solving a polynomial number of the corresponding decision problems. If the optimization problem is easy, so is its related decision problem. Conversely, if the decision problem is hard so is the corresponding optimization problem. So, even though it restricts to decision problems, the theory of $\mathcal{NP}$-completeness applies much more widely.

For more insight on computational complexity we refer to the textbooks of Garey and Johnson [23], Papadimitriou and Steiglitz [60], and Johnson and Papadimitriou [38].

1.3 Polyhedral theory

We denote the Euclidean linear space of dimension n by $\mathbb{R}^n$. If E is some finite set we denote by $\mathbb{R}^E$ the set of real vectors having $|E|$ components which are indexed by the members of E. We assume vectors to be column vectors and we denote the transposition of a vector x by x^T.

Let $m > 0$ and $x_1, ..., x_m \in \mathbb{R}^n$. A vector $y \in \mathbb{R}^n$ is a *linear combination* of vectors x_i, $i = 1, ..., m$ if $y = \sum_{i=1}^m \lambda_i x_i$ for some $\lambda_1, ..., \lambda_m \in \mathbb{R}$. If, moreover, $\sum_{i=1}^m \lambda_i = 1$, y is an *affine combination* of vectors x_i. An affine combination with $\lambda_i \geq 0$ for $i = 1, ..., m$, is a *convex combination*.

Given a set $S \subseteq \mathbb{R}^n$, the *convex hull* of S, denoted by $conv(S)$, is the set of all points that are convex combination of points in S. The *affine hull* of S, *aff(S)*, is defined in a similar way.

A set of points $S = \{x_1, ..., x_m\} \subseteq \mathbb{R}^n$ is *linearly (affinely) independent* if the unique solution of $\sum_{i=1}^m \lambda_i x_i = 0$ ($\sum_{i=1}^m \lambda_i x_i = 0$ and $\sum_{i=1}^m \lambda_i = 0$) is $\lambda_i = 0$, $i = 1, ..., m$; otherwise it is *linearly (affinely) dependent*. For sets S with at least two elements, linear (affine) independence means that no $x \in S$ can be represented as a linear (affine) combination of the vectors in $S \backslash \{x\}$. Linear independence implies affine independence, but the converse is not true.

Theorem 2 *Let $S = \{x_0, x_1, ..., x_m\}$ be a set of $m+1$ points in $\mathbb{R}^n$. Then the following statements are equivalent:*

1. The points in S are affinely independent.

2. For any $w \in \mathbb{R}^n$, the points $\{x_i - w : i = 0, 1, ..., m\}$ are affinely independent.

3. The m points $\{x_i - x_0 : i = 1, ..., m\}$ are linearly independent.

If $a \in \mathbb{R}^n \backslash \{0\}$ and $a_0 \in \mathbb{R}$ then $\{x \in \mathbb{R}^n : a^T x = a_0\}$ is called a *hyperplane*. A hyperplane defines the *halfspace* $\{x \in \mathbb{R}^n : a^T x \leq a_0\}$. A *polyhedron* P is defined as the intersection of a finite number of halfspaces or, equivalently, as the set of points that satisfy a finite number of linear inequalities. More precisely, P is a polyhedron if there exists a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$ such that $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. Note that some inequalities may actually be equations. To emphasize this matter we write $P = \{x \in \mathbb{R}^n : Ax \leq b, Bx = d\}$.

A *polytope* is a bounded polyhedron, that is, $P \subseteq \mathbb{R}^n$ is a polytope if it is a polyhedron and there exist $l, u \in \mathbb{R}^n$ such that $l \leq x \leq u$ for all $x \in P$. A polytope can also be defined as the convex hull of a finite set of points.

The *dimension* of a polyhedron P, $dim(P)$, is the maximum number of affinely independent points in P minus one. $P \subseteq \mathbb{R}^n$ is said to be *full-dimensional* if $dim(P) = n$.

An inequality $a^T x \leq a_0$ is said to be *valid* for a polyhedron $P \subseteq \mathbb{R}^n$ if $P \subseteq \{x \in \mathbb{R}^n : a^T x \leq a_0\}$. If $a^T x \leq a_0$ is a valid inequality for P then the set $F = P \cap \{x : a^T x = a_0\}$ defines a *face* of P. If $F \neq \emptyset$ and $F \neq P$ then F is called a *proper face*, and it is said to be the face *induced* by $a^T x \leq a_0$. The empty set and P itself are called *improper faces* of P. If $a^T x \leq a_0$ induces a face F of P, $\overline{a}^T x \leq \overline{a}_0$ induces another face $\overline{F}$ of P, and $F \subset \overline{F} \subset P$ then it is said that $a^T x \leq a_0$ *dominates* $\overline{a}^T x \leq \overline{a}_0$, or equivalently, $a^T x \leq a_0$ is *stronger* than $\overline{a}^T x \leq \overline{a}_0$.

The most important faces of a polyhedron are those proper faces which are minimal or maximal with respect to set inclusion. If F is a face whose dimension is zero, then the unique element $v \in F$ is a *vertex* of P. Vertices have the property that they cannot be represented by convex combinations of other points of the polyhedron. If P is a polytope, then we have $P = conv(V)$ where V is the set of vertices of P. A *facet* is a maximal proper face. If

$F = P \cap \{x \in \mathbb{R}^n : a^T x = a_0\}$ is a facet we call $a^T x \leq a_0$ a *facet defining inequality* for P.

A linear system with both inequalities and equalities that determines a polyhedron P is said to be *minimal* when:

(i) no inequality can be set to equality without reducing the polyhedron, and

(ii) no inequality or equality can be removed without changing the polyhedron.

The importance given to the inequalities defining facets is based on the following result.

Theorem 3 *Let $P = \{x \in \mathbb{R}^n : Ax \leq b, Bx = d\}$. Then $Ax \leq b, Bx = d$ is a minimal lineal system for P if and only if*

1. no equality in $Bx = d$ is a linear combination of the other equations of that system, and

2. each inequality in $Ax \leq b$ defines a facet of P and conversely.

The following theorem gives a further characterization of the facet defining inequalities. This result shows the two main methods used to proof that an inequality defines a facet of a polyhedron.

Theorem 4 *Let F be a proper face of $P = \{x \in \mathbb{R}^n : Ax \leq b, Bx = d\}$ and let h be the number of equations in this system. Then the following statements are equivalent:*

1. F is a facet of P.

2. $\dim(F) = \dim(P) - 1$.

3. If $F = \{x \in P : a^T x = a_0\} = \{x \in P : \overline{a}^T x = \overline{a}_0\}$, then there exist $\delta \geq \mathbb{R}^+$ and $\lambda \in \mathbb{R}^h$ such that $\overline{a} = \delta a + \lambda B$ and $\overline{a}_0 = \delta a_0 + \lambda d$.

For more details on polyhedral theory we refer to the textbooks of Nemhauser and Wolsey [57], Schrijver [64] and Wolsey [71].

1.4 Polyhedral combinatorics

Let E be a finite set, and let $\mathcal{F}$ be a finite family of subsets of E called feasible sets. Also let $w : E \longrightarrow \mathbb{R}_+$ be a cost function defined on E. Define $w(F) = \sum_{e \in F} w_e$ for each $F \in \mathcal{F}$. We then call the problem

$$\max\{w(F) : F \in \mathcal{F}\}$$

a *combinatorial optimization problem.*

A classical example of combinatorial optimization problem is the *travelling salesman problem* (TSP). Given a set of cities and the distances between each pair of them, the TSP consists of finding the shortest possible tour that visits each city exactly once. This $\mathcal{NP}$-hard problem is perhaps the most famous in Operational Research.

We will now associate polytopes with combinatorial optimization problems. For a subset $F \subseteq E$ we define the *incidence vector* $x^F \in \mathbb{R}^E$ as follows

$$x_e^F = \begin{cases} 1, & \text{if } e \in F \\ 0, & \text{if } e \notin F \end{cases} .$$

The polyhedron associated with $\mathcal{F}$ is defined as

$$P_{\mathcal{F}} = conv\{x^F \in \mathbb{R}^E : F \in \mathcal{F}\},$$

that is, $P_{\mathcal{F}}$ is the convex hull of the incidence vectors of feasible sets. The combinatorial optimization problem can now be solved via the linear programming problem

$$\max\{w^T x : x \in P_{\mathcal{F}}\} \tag{1.1}$$

since every solution of (1.1) is a vertex of $P_{\mathcal{F}}$ and therefore a feasible set.

Suppose that there is a matrix $A \in \mathbb{R}^{m \times E}$ and a vector $b \in \mathbb{R}^m$ such that

$$P_{\mathcal{F}} = \{x \in I\!R^E : Ax \leq b\}.$$

Then solving the optimization problem is equivalent to finding the optimum vertex solution of the linear problem (LP)

$$\begin{array}{ll} \max & w^T x \\ s.t. & Ax \leq b. \end{array}$$

A first difficulty in using this approach is that the number m of inequalities defining $P_{\mathcal{F}}$ may be very large even for very small optimization problems. Therefore, even if the complete description of $P_{\mathcal{F}}$ is known, it is in general not possible to list all inequalities and to use the simplex algorithm for finding an optimal vertex. A *cutting plane algorithm* proceeds as shown below:

Cutting plane algorithm

(1) Create an initial LP with only some inequalities $\overline{A}x \leq \overline{b}$ of the system $Ax \leq b$.

(2) Solve the LP and let an optimal solution be x^*.

(3) If $x^* \in \mathcal{F}$, stop: it is an optimal solution.

(4) Otherwise, identify an inequality of the system $Ax \leq b$ which is violated by x^*.

(5) Add this inequality to $\overline{A}x \leq \overline{b}$ and go to step (2).

end

The central point of this methodology is the identification of violated inequalities if the current x^* is not a feasible incidence vector. This is the so-called *separation problem*, which, given a point $x^* \in I\!R^E$ and a polyhedron $P_{\mathcal{F}}$, consists in showing that x^* satisfies all the inequalities defining $P_{\mathcal{F}}$ or giving one of them that is violated by x^*. The importance of this problem is due to the following result, given by Grötschel et al. [30]:

Theorem 5 *A combinatorial optimization problem can be solved in polynomial time if and only if the separation problem for the associated polytope can be solved in polynomial time.*

A second difficulty arises from the fact that generally, when trying to solve hard combinatorial optimization problems we do not know, or we cannot cope with, a complete linear description of $P_{\mathcal{F}}$, but just a partial one. In these cases the cutting plane approach may fail in step (4), giving as result a non feasible (i.e., fractional) point x^* for which no violated inequality can be found. At this point we can resort to the classical principle of Branch-and-Bound. Note that $w^T x^*$ gives and upper bound on the objective function value of all feasible solutions of the subproblems. A Branch-and-Bound algorithm works as follows:

Branch-and-Bound algorithm

(1) Initialize the list of active subproblems with the original problem.

(2) If the list of active subproblems is empty, stop: the best feasible incidence vector found so far is optimal.

(3) Otherwise, choose some subproblem from the list of active problems and 'solve' it as follows:

- find and optimal incidence vector for the subproblem, or

- prove that the subproblem has not feasible solution, or

- prove that there is no feasible incidence vector for the subproblem that has a better objective function value than the best feasible incidence vector known so far, or

- split the subproblem into further subproblems and add them to the list of active problems, if none of the above is possible.

(4) Go to step (2).

end

The term *Branch-and-Cut* was first used in Padberg and Rinaldi [59] for an algorithm to solve the TSP. It applies to those Branch-and-Bound algorithms that use cutting planes to solve the current problem in step (3).

1.5 Multicriteria optimization

Generally, in multicriteria problems an optimal solution does not exist because the objectives are in conflict. The concept of optimal solution is then replaced by that of *efficient solution* (also *noninferior* or *Pareto optimal*). An efficient solution is one for which an improvement in one of the objectives will necessarily result in a loss in at least one of the other objectives.

The efficient set of multiobjective integer programs may be very large and may even grow exponentially with problem size, making it vary hard to compute. Because of the computational burden associated with generating the entire efficient set, it is usually satisfactory to generate an approximation of the efficient solution set. Two main approximated solution techniques for multiobjective problems formulated as integer programs are the *constraint method* and the *utility function method*.

The constraint method converts all but one of the objectives into constraints so as to optimize with respect to only a single objective. For example, for a bicriterium problem involving min cost and max accessibility the two possible constrained formulations would be (1) minimizing cost subject to an accessibility constraint, or (2) maximizing accessibility subject to a cost constraint.

The utility function method consists of combining the different objectives into a unique objective function to be optimized. A very often used technique is the *weighting method*, in which every objective is multiplied by a non-negative weight. Then, the weighted objectives are summed to form a weighted-sum objective function which is minimized (or maximized) to obtain an efficient solution. This process is repeated with a series of weights to generate an approximation of the efficient set. The endpoints of the efficient set are obtained by minimizing (or maximizing) the original objective functions individually. This methodology provides only efficient solutions that are on the convex hull of the solution set.

For more details on muticriteria optimization we refer to Steuer [65].

Chapter 2

Path, Tree and Cycle location

Extensive facilities are structures that are too large to be considered as single points. In Section 2.2 we propose integer programming formulations for biobjective location problems of extensive facilities on undirected graphs, which can also be considered as routing problems, under min-cost, max-cover and min-distsum criteria. Regarding the shape, we have restricted our attention to path, tree and cycle shaped structures. Our aim is to give a generic way of making integer programming formulations of these kind of problems. Sections 2.3 to 2.7 review the existing literature on this topic, and try to establish the complexity of all problems.

2.1 Introduction

In this chapter, we study biobjective location problems of one extensive facility on a graph, under min-cost and max-accessibility criteria. We consider only those problems involving desirable facilities. In fact, we focus on three well defined objectives:

- *(o1)* min-cost: minimize the total cost of the facility;
- *(o2)* max-cover: maximize the total demand covered by the facility;
- *(o3)* min-distsum: minimize the total distance from the facility to the vertices of the graph;

This leads to three possible biobjective problems: *(min-cost, max-cover)*, *(min-cost, min-distsum)* and *(max-cover, min-distsum)*. Moreover, for $i < j$, each biobjective problem *(oi, oj)* may be approached in three different ways:

- optimize a linear combination of the objectives: $\lambda_1(oi) + \lambda_2(oj)$,
- optimize *(oi)* subject to a restriction on *(oj)*, or
- optimize *(oj)* subject to a restriction on *(oi)*.

We do not consider the other classical criteria in location analysis: the *min-max* criterion. The reason is that we think that max-cover and min-distsum are better suited than the min-max criterion to the generic goal of maximizing accessibility. Indeed, the latter intends to minimize the damage caused to the furthest demand point and is more appropriate in emergency service location problems such as that of fire stations.

As to the facilities to be located, we restrict ourselves to the five following types of extensive facilities:

- Paths with fixed end points.
- Paths without fixed end points.
- Trees.
- Cycles with a fixed origin.
- Cycles without a fixed origin.

We consider them as the most representative. Path and cycle structures are common in routing applications, and the tree is commonly associated to network design problems since it is the least costly connected structure.

2.2 Notation and formulations

Let $G = (V, E)$ be an undirected graph or network. Let c_e denote the cost of an edge $e = [v_i, v_j] \in E$. As to the demand structure, we will consider that

demand arises only at the vertices. Let $w_i \geq 0$ be the demand associated with $v_i \in V$. Let $W = \sum_{v_i \in V} w_i$ be the total demand in the graph. If $w_i = 1$ for all $v_i \in V$ we say the graph is *unweighted*; otherwise the graph is *weighted*.

When considering covering, for a given value $r \geq 0$ we define $S_i = \{v_j \in V : d(v_i, v_j) \leq r\}$ as the set of vertices that can indirectly cover vertex v_i. If $r = 0$ then we talk of *direct covering*, a special case of *indirect covering*.

To derive integer programming formulations we need the additional notations given in Table 2.1.

$$x_e = \begin{cases} 1 & \text{if edge } e \text{ is in the solution,} \\ 0 & \text{otherwise;} \end{cases}$$

$$z_i = \begin{cases} 1 & \text{if node } v_i \text{ is indirectly covered,} \\ 0 & \text{otherwise;} \end{cases}$$

$$y_{ij} = \begin{cases} 1 & \text{if } v_i \text{ is assigned to } v_j, \\ 0 & \text{otherwise.} \end{cases}$$

Table 2.1: Decisional variables

Note that with this notation, $y_{ii} = 1$ means that vertex v_i is in the solution.

Given a nonnegative constant K, we can define cost, coverage and distsum either as objective functions to be optimized or as constraints to be satisfied in the following way. When considered as objective function, the cost can be written as

$$\text{minimize} \sum_{e \in E} c_e x_e, \tag{2.1}$$

while when considered as a constraint, it is expressed as

$$\sum_{e \in E} c_e x_e \leq K. \tag{2.2}$$

Similarly, for the coverage, the objective function is

$$\text{maximize} \sum_{v_i \in V} w_i z_i, \tag{2.3}$$

if indirect covering is considered and

$$\text{maximize} \sum_{v_i \in V} w_i y_{ii}, \tag{2.4}$$

if the demand is to be cover directly. The corresponding constraints are

$$\sum_{v_i \in V} w_i z_i \geq K \tag{2.5}$$

for the indirect covering, and

$$\sum_{v_i \in V} w_i y_{ii} \geq K \tag{2.6}$$

for the direct one.

If $K = W$, that is, if all the demand of the graph must be covered, we will talk of *total* covering of the demand, while if $K < W$ we will talk of *partial* covering of the demand. Note that in an unweighted graph, a constraint on the direct covering of the facility is a constraint on the number of vertices it contains. The most general form of covering is indirect partial weighted covering. In all cases concerning covering, the following constraints must be added to the model:

$$y_{ii} \leq z_i, \qquad \forall v_i \in V \tag{2.7}$$

$$z_i \leq \sum_{v_j \in S_i} y_{jj}, \qquad \forall v_i \in V. \tag{2.8}$$

Constraints (2.7) mean that direct covering implies indirect covering. Constraints (2.8) ensure that a vertex v_i is covered only if some vertex $v_j \in S_i$ is in the solution. Finally, distsum yields the objective function

$$\text{minimize} \sum_{v_i} \sum_{v_j} w_j d(v_j, v_i) y_{ji}, \tag{2.9}$$

and the constraint

$$\sum_{v_i} \sum_{v_j} w_j d(v_j, v_i) y_{ji} \leq K. \tag{2.10}$$

Here again, the following additional constraints must be included in the model:

$$\sum_{v_j \neq v_i} y_{ij} + y_{ii} = 1, \qquad \forall v_i \in V, \tag{2.11}$$

$$y_{ij} \leq y_{jj}, \qquad \forall v_i \in V, \forall v_j \in V. \tag{2.12}$$

Constraints (2.11) force each vertex v_i to be either in the solution or assigned to another vertex v_j. Constraints (2.12) mean that a vertex v_i can be assigned to a vertex v_j only if v_j is in the solution.

We now present constraints defining the five different location structures we consider.

- Paths with fixed end points v_1 and v_n.

$$\sum_{e \in \delta(i)} x_e = \begin{cases} 2y_{ii}, & v_i \in V \backslash \{v_1, v_n\} \\ 1, & v_i \in \{v_1, v_n\} \end{cases} \tag{2.13}$$

$$\sum_{e \in \delta(S)} x_e \geq y_{kk}, \quad \forall S \subset V,\ v_1 \notin S \text{ or } v_n \notin S, v_k \in S, \quad 2 \leq \mid S \mid \leq n-1 \tag{2.14}$$

$$y_{11} = 1 \tag{2.15}$$

$$y_{nn} = 1 \tag{2.16}$$

- Paths without fixed end points.

$$\sum_{e \in E} x_e = \sum_{v_i \in V} y_{ii} - 1 \tag{2.17}$$

$$\sum_{e \in \delta(S)} x_e \geq y_{kk} + y_{hh} - 1, \quad \forall S \subset V, v_k \in S,\ v_h \notin S, \quad 2 \leq \mid S \mid \leq n-1 \tag{2.18}$$

$$\sum_{e \in \delta(i)} x_e \leq 2y_{ii}, \quad \forall v_i \in V. \tag{2.19}$$

- Trees.

$$\sum_{e\in E} x_e = \sum_{i\in V} y_{ii} - 1$$
$$\sum_{e\in\delta(S)} x_e \geq y_{kk} + y_{hh} - 1, \quad \forall S \subset V, v_k \in S,\ v_h \notin S,$$
$$2 \leq \mid S \mid \leq n-1.$$

- Cycles with a fixed origin v_1.

$$\sum_{e\in\delta(i)} x_e = 2y_{ii}, \quad \forall v_i \in V \tag{2.20}$$
$$\sum_{e\in\delta(S)} x_e \geq 2y_{kk}, \quad \forall S \subset V,\ v_1 \notin S, v_k \in S, \quad 2 \leq \mid S \mid \leq n-1 \tag{2.21}$$
$$y_{11} = 1$$

- Cycles without a fixed origin.

$$\sum_{e\in\delta(i)} x_e = 2y_{ii}, \quad \forall v_i \in V$$
$$\sum_{e\in\delta(S)} x_e \geq 2(y_{kk} + y_{hh} - 1), \quad \forall S \subset V, v_k \in S,\ v_h \notin S, \quad 2 \leq \mid S \mid \leq n-1. \tag{2.22}$$

Constraints (2.13) imply that each internal vertex of the path has degree two and each extremity has degree one. Constraints (2.14) are subtour elimination constraints. Constraints of type (2.15) and (2.16) force a given vertex to be in the solution. Constraints (2.17) and (2.18) define a tree facility, since constraint (2.17) requests a number of edges equal to the number of vertices minus one in the solution, and constraints (2.18) imply connectivity. If we add (2.19) we define a path, since (2.19) forces the degree of each vertex in the facility to be at most equal to two. Constraints (2.20) impose that each

vertex of the solution has degree two. Subtour elimination constraints (2.21) and (2.22) differ according to whether the cycle has a fixed origin or not.

Conveniently combining all these constraints, those used to define cost, covering and distsum, and those used to model the facilities, we arrive at generic integer programming formulations for all the locations problems on undirected graphs considered in this chapter. Note that partial edges are not allowed. We now give some examples.

Example 2.1: In a general graph, find a minimum cost path from a vertex v_1 to a vertex v_n that indirectly covers a total demand of value at least K. This problem is formulated as follows:

$$\begin{aligned}
&\text{minimize } \sum_{e\in E} c_e x_e \\
&\text{s.t.} \\
&\sum_{v_i\in V} w_i z_i \geq K \\
&z_i \leq \sum_{v_j\in S_i} y_{jj}, && \forall v_i \in V \\
&y_{ii} \leq z_i, && \forall v_i \in V \\
&\sum_{e\in\delta(i)} x_e = \begin{cases} 1, & v_i \in \{v_1,\, v_n\} \\ 2y_{ii}, & \text{otherwise} \end{cases} \\
&\sum_{e\in\delta(S)} x_e \geq y_{kk}, && \forall S \subset V, v_k \in S,\ v_1 \notin S \text{ or } v_n \notin S, \\
& && 2 \leq \mid S \mid \leq n-1 \\
&y_{11} = 1 \\
&y_{nn} = 1 \\
&x_e, y_{ii}, z_i \in \{0,1\} && \forall v_i \in V, \forall e \in E.
\end{aligned}$$

Example 2.2: In a graph, find a tree with cost less than or equal to K that minimizes the sum of distances to the vertices. The corresponding formulation is:

$$
\begin{aligned}
&\text{minimize} \sum_{v_i} \sum_{v_j} w_j d(j,i) y_{ji} \\
&\text{s.t.} \\
&\sum_{e \in E} c_e x_e \leq K \\
&\sum_{v_j \neq v_i} y_{ij} + y_{ii} = 1, && \forall v_i \in V \\
&y_{ij} \leq y_{jj}, && \forall v_i, v_j \in V \\
&\sum_{e \in E} x_e = \sum_{v_i \in V} y_{ii} - 1 \\
&\sum_{e \in \delta(S)} x_e \geq y_{kk} + y_{hh} - 1, && \forall S \subset V, v_k \in S,\ v_h \notin S, \\
& && 2 \leq \mid S \mid \leq n-1 \\
&x_e, y_{ij} \in \{0,1\}, && \forall v_i, v_j \in V, \forall e \in E.
\end{aligned}
$$

Example 2.3: Locate a tour on a general graph, passing through vertex v_1, minimizing the sum of distances to the vertices and imposing a constraint on the facility cost. This can be formulated as:

$$
\begin{aligned}
&\text{minimize} \sum_{v_i} \sum_{v_j} w_j d(j,i) y_{ji} \\
&\text{s.t.} \\
&\sum_{e \in E} c_e x_e \leq K \\
&\sum_{v_j \neq v_i} y_{ij} + y_{ii} = 1, && \forall v_i \in V \\
&y_{ij} \leq y_{jj}, && \forall v_i, v_j \in V \\
&\sum_{e \in \delta(i)} x_e = 2 y_{ii}, && \forall v_i \in V \\
&\sum_{e \in \delta(S)} x_e \geq 2 y_{kk}, && \forall S \subset V, v_k \in S,\ v_1 \notin S, \\
& && 2 \leq \mid S \mid \leq n-1 \\
&y_{11} = 1 \\
&x_e, y_{ij} \in \{0,1\}, && \forall v_i, v_j \in V, \forall e \in E.
\end{aligned}
$$

2.3 Paths with fixed end points

In a tree, the problem of finding an optimal path connecting two given points is trivial since there is only one such a path. On the other hand, in a general graph the nine biobjective problems considered are $\mathcal{NP}$-hard, since for each one there exits a reduction from either the problem of finding a Hamiltonian path or the problem of finding a shortest Hamiltonian path between two fixed end vertices (see Hakimi et al. [33] for the *min-distsum s.t. min-cost* case).

We now review the existing literature on biobjective location of path-shaped facilities on general graphs, considering min-cost, max-cover and min-distsum.

(Min-cost, max-cover)

Current et al. [14] introduce the biobjective problem of locating a path from a predetermined starting vertex to a predefined terminus vertex in order to minimize its length and maximize the covered demand. They call the problem the Maximum Covering Shortest Path problem (MCSP) when indirect covering of the demand is allowed, and the Maximum Population Shortest Path problem (MPSP) when the demand has to be directly covered, and present integer programming formulation for both cases. They use the weighting method to generate efficient solutions, and solve the problem by relaxing the integrality and subtour elimination constraints. If noninteger solutions appear, then Branch-and-Bound is applied. When subtours occur, the necessary subtour breaking constraints are added and the problem is solved again. A sample MPSP problem with 15 vertices and 34 edges is given.

Min-cost s.t. max-cover

The problem with indirect total weighted covering has been studied by Current et al. [12] and [13]. It was first introduced in Current et al. [12] with the name of Shortest Covering Path Problem (SCPP) and presented as a synthesis of the Location Set Covering Problem and the Shortest Path Problem. In that paper, an integer programming formulation of the problem is given and two particular instances, with 20 vertices and 190 edges and 15 vertices and 33 edges respectively, are solved by relaxing the integrality and subtour elimination constraints. Branch-and-Bound is applied when noninteger solutions appear and subtour elimination constraints are added when necessary.

Current et al. [13] show that this problem is $\mathcal{NP}$-hard since, if the instance where each vertex is only covered by itself (i.e., direct covering) is considered, then the problem consists of finding the shortest Hamiltonian Path from between the two fixed end vertices.

They give an integer programming formulation for the problem on di-

rected graphs and propose a heuristic and an exact solution method. The heuristic is based upon a Lagrangian relaxation of the problem in which the set of constraints imposing covering restrictions are dualized. This Lagrangian relaxation has the structure of a shortest path problem with additional subtour elimination constraints. A subgradient optimization method is used to find multipliers, discarding those that lead to negative cycles in the shortest path subproblems. The subproblems can thus be solved using Floyd's algorithm (Floyd [22]). The exact method is based upon a Branch-and-Bound procedure which uses the bounds generated by the heuristic.

Both the heuristic and the Branch-and-Bound algorithms are tested on 160 randomly generated graphs with up to 80 vertices and 8000 edges. The heuristic solves optimally up to the 84% of all those problems. The average gap between the heuristic solution and the Lagrangian relaxation solution is smaller than 1%, with a maximum of 17%. Moreover, most problems are solved in less than 1000 seconds on a CDC CYBER 76 computer, although the largest ones require almost one hour.

(Min-cost, min-distsum)

Current et al. [15] provide an algorithm (MONET) that generates efficient solutions to the bicriterion problem called the Median Shortest Path Problem (MSPP). MONET is an enumeration algorithm based on the k-shortest path problem. Its complexity is $O(kn^2 + n^3 + k^2)$ on complete graphs and $O(km \log n + mn \log_{2+m/n} n + k^2)$ on sparse graphs. Compared with the weighting method, MONET seems to be a good and quick algorithm to generate efficient solutions for medium size instances of the MSPP. Moreover, it identifies a series of efficient solutions that the weighting method does not find because they are not on the convex hull of the solution set.

Avella and Sforza [1] and [2] study the Median Path Problem (MPP), which is a version of the problem consisting of minimizing the sum of the two objectives, cost and sum of distances. They propose an integer programming formulation, based on a new class of inequalities which are proved to be facet defining for the MPP polytope. To solve the problem they develop both a Branch-and-Cut algorithm and a Lagrangian heuristic.

2.4 Paths without fixed end points

The number of paths on a tree is $O(n^2)$ and therefore all problems consisting of locating a path on a tree are polynomial, unless there is a nonpolynomial objective function to be evaluated for each path. For example, the nine problems we consider can be solved in $O(n^3)$ operations. However, on general graphs all these problems are $\mathcal{NP}$-hard, since it is possible to find a reduction from either the Hamiltonian path or to the shortest Hamiltonian path problem to each of them (the proof for *min-distsum s.t. min-cost* is provided in Hakimi et al. [33]).

Min-distsum s.t. min-cost

On tree graphs, Minieka and Patel [52] consider the problem of finding the core or path median of length l, when partial edges are allowed in the solution. They did not succeed in fully characterizing the solution, and the problem of developing an efficient algorithm remained unsolved until Minieka [51] proposed a solution method whose complexity depends on the number of leaves of the tree. In fact, the algorithm consists of, for every pair of leaves v_i, v_j with $d(v_i, v_j) \geq l$, moving a path of length l along the unique path from v_i to v_j until a certain condition becomes true, choosing then the best among all those candidates paths. Since the number of leaves of a tree is $O(n)$ and the best path for a given pair of leaves can be found in linear time, Minieka's algorithm has $O(n^3)$ complexity.

On general graphs, Richey [62] studies the problem of finding the path which minimizes the sum of distances to the vertices, provided its length is not larger than a given value. The solution may contain partial arcs. Richey shows that the problem is $\mathcal{NP}$-hard in general, since the Hamiltonian Path problem can be reduced to it, but when the constraint on the length is an equality it can be solved for a class of graphs (series-parallel graphs) using pseudo-polynomial algorithms. If partial edges are not allowed, the problem is also $\mathcal{NP}$-hard (see Hakimi et al. [33]).

2.5 Trees

The constrained formulations of the biobjective problems *(min-cost, max-cover),* with direct partial weighted covering of the demand, and *(min-cost, min-distsum)* are $\mathcal{NP}$-hard even on tree graphs. The proof for *(min-cost, min-distsum)* is given by Hakimi et al. [33]. Here is the proof for *(min-cost, max-cover).*

Consider the decision version of the direct partial weighted *max-cover s.t. min-cost* problem on a tree graph:

Problem 2.1: Given a tree T and positive numbers K and l_o, is there a subtree T_o with cost less than or equal to l_o and such that the demand covered by T_o is at least K?

Theorem 6 *Problem 2.1 is $\mathcal{NP}$-complete.*

Proof.

To prove that this problem is $\mathcal{NP}$-complete we reduce the Partition Problem to it. Given a sequence $a_1, a_2, ..., a_n$ of positive integers with $\sum_{i=1}^{n} a_i = S$, the Partition Problem asks whether there is a subset $I \subset \{1, ..., n\}$ with $\sum\{a_i : i \in I\} = S/2$. We build a star tree as indicated in Figure 2.1, in which the demand of each vertex v_i, $i = 1, 2, ..., n$, is $w_i = a_i$, the demand of vertex v_0 is $w_0 = 0$, and the length of each edge $[v_0, v_i]$ is a_i. Let it be $W = \sum_{i=1}^{n} w_i$, and let take $l_o = W/2$ and $K = W/2$. Then it is easy to see that a solution to our problem exists for the graph of Figure 2.1 if and only if there is a subset of vertices with total demand equal to $W/2$, that is, if and only if the Partition Problem has a solution. As the Partition Problem is $\mathcal{NP}$-complete , so is our problem. ■

The same transformation can be used to prove the $\mathcal{NP}$-hardness of the *min-cost s.t. max-cover* problem. Furthermore, notice that the problem with direct partial weighted covering is a special case of the general problem with indirect partial weighted covering, which is therefore $\mathcal{NP}$-hard also.

On the other hand, the three formulations involving *max-cover* and *min-distsum* are trivial on tree graphs (they have the whole tree as solution), and polynomial on general graphs (they reduce to finding a spanning tree).

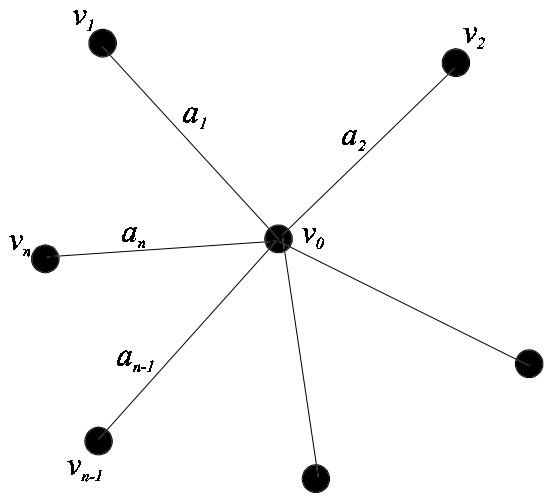

Figure 2.1: A star graph

λ_1(min-cost) + λ_2(max-cover)

The problem of minimizing a linear combination of the *min-cost* and *max-cover* objectives is $\mathcal{NP}$-hard since the Steiner Tree Problem, which is known to be $\mathcal{NP}$-hard, is easily reduced to it. The Steiner Tree Problem is the consists of finding a minimum cost subtree of a graph containing all vertices in S, where S is a subset of V. For the reduction, consider direct covering of the demand and set λ_1 to a small number and the demand of each vertex to $w_i = M$ for all $v_i \in S$, being M a very large number, and $w_i = 0$ otherwise. With this transformation, solving the λ_1*(min-cost)* + λ_2*(max-cover)* problem is equivalent to solving the Steiner Tree Problem.

The review of the literature reveals that this problem has been studied by several authors, but only on tree graphs. It seems that it was first introduced by Hutson and ReVelle [36], who proposed an integer programming bicriterion formulation for the case with direct partial weighted covering of the demand and use the weighting method to approximate the efficient set. The optimal integer solution to the relaxed weighted problem (found using Branch-and-Bound if necessary) is an efficient solution for the biobjective problem. They also considered the case where some edges are required to be in the solution, that is, the case of 'adding to an existing structure'. Hutson and ReVelle [37] used the same approach to solve the problem with indirect

covering of the demand.

Tamir [66] shows that the weighting method used by Hutson and ReVelle [36] can be implemented in $O(n^2)$ time using dynamic programming. Church and Current [10] propose a new integer formulation and another $O(n^2)$ time algorithm called BUCS (Building a Covering Subtree), that finds the subtree minimizing a linear combination of min-cost and max-cover with given weights. BUCS is a constructive algorithm which builds the best subtree from a given edge. Then the best of those subtrees is chosen as the solution. BUCS can also be applied to indirect covering of the demand when partial arcs are allowed, and to the problem of adding to an existing structure.

Finally, Kim et al. [40] give $O(n)$ and $O(n \log^2 n)$ time dynamic programming algorithms for solving the linear combination formulation with direct and indirect covering of the demand, respectively. The algorithms can be seen as special cases of a method developed to solve a more general problem. The authors take the sum of the facility length and of the demand covered as the function to minimize.

Min-cost s.t. max-cover

On tree graphs, the direct total covering problem is trivial since the solution is the whole tree. Hutson and ReVelle [37] propose an integer programming formulation of the problem with indirect total covering. The solution method they give extends reduction techniques used by Toregas and ReVelle in [68] and [69] to solve the Location Set Covering Problem on tree graphs. Kim et al. [43] consider a formulation with indirect total covering which differs from the one in Hutson et al. [37] since there are now different covering distances r_i for each $v_i \in V$, and partial edges are allowed in the solution. The authors present an $O(n)$ time exact method based on the algorithm for the Set Covering Problem. Maffioli [49] studies the problem with direct partial unweighted covering, that is, the problem of finding the minimum cost subtree with at least K vertices. He gives an $O(K^2 n)$ dynamic programming exact algorithm. Ehrgott et al. [18] give a polynomial time algorithm for the version of this problem that looks for a subtree with exactly K vertices.

On general graphs, Kim et al. [42] formulate the Subtree r-Cover Problem, which consists of finding the minimal length subtree of a graph that indirectly covers all the vertices (indirect total weighted covering). A vertex v_i is said to

be covered by the facility if it is within a given distance r_i from it, and partial arcs are allowed in the solution. The authors show that this problem is $\mathcal{NP}$-hard, and give a pseudo-polynomial algorithm that exploits the structure of the graphs and is polynomial on some of them (e.g., in cactus graphs). Ehrgott et al. [18] study the problem of finding the minimum length subtree with exactly K vertices in a general unweighted graph. They show this problem is $\mathcal{NP}$-hard and present different heuristic approaches based on spanning tree and shortest path methods and on the exact algorithm for solving the problem in polynomial time on tree graphs.

Max-cover s.t. min-cost

The problem of maximizing the demand covered subject to the constraint that the cost of the tree is less than or equal to a given value l has only been treated on tree graphs. It is trivial when l is at least equal to the total cost of the tree. Tamir [67] proposes a $(1+\epsilon)$-approximation method for solving the *min-distsum s.t. min-cost* problem with complexity $O(n^2/\epsilon)$. He shows that the method can be easily modified to solve the problem with indirect covering of the demand. Another related paper is that of Church and Current [10], who apply Branch-and-Bound to solve the problem of minimizing a linear combination of objectives *min-cost* and *max-cover*, with direct covering of the demand, subject to a budget (cost) constraint.

λ_1(min-cost) + λ_2(min-distsum)

On tree graphs, Kim et al. [40] and Kim et al. [41] present two different methods with $O(n)$ complexity to solve the Median Subtree Location Problem, consisting of minimizing the sum of the facility cost and sum of distances. The method proposed by Kim et al. [41] is in some way similar to Goldman's algorithm for the median of a tree given in Goldman [29]. In the second paper, Kim et al. [40] propose a dynamic programming method.

On general graphs, Kim el al. [41] optimize the facility cost plus the sum of distances, and call the problem the Median Subtree Problem. They show that the problem is $\mathcal{NP}$-hard and give a solution method that uses the concept of block of a graph. The method reduces the original graph G to a graph G'. This new graph G' is such that it contains a feasible

subtree T having nonempty intersection with each block of G'. It is then shown that the problem can be decomposed into solving, in linear time, a Median Subtree Problem on each spanning tree within each block of G'. This leads to a pseudo-polynomial algorithm, although it is polynomial when the underlying graph is a cactus.

Min-distsum s.t. min-cost

On tree graphs, Tamir [67] gives an algorithm that generates a $(1+\epsilon)$-approximation in $O(n^2/\epsilon)$ time. The algorithm is based on a standard application of the interval partitioning method suggested by Sahni [63] and makes use of dynamic programming techniques. Minieka [51] formulates the problem with an equality constraint on the cost of the facility. Note that in this case the solution may contain partial arcs. He gives a polynomial time solution method which, starting from the set M of medians of the tree, builds up a solution subtree T. This method is based on the fact that, for any feasible length l, there is a minimum distsum subtree of size l, T_l, satisfying that $T_l \cap M \neq \emptyset$. On general graphs, Richey [62] gives a pseudo-polynomial algorithm that solves the problem when partial edges are allowed.

2.6 Cycles with an origin

All nine problems we consider under this heading are $\mathcal{NP}$-hard since either the Hamiltonian cycle problem or the Traveling Salesman Problem reduces to each of them.

Note also the relation existing between these problems and those involving the location of a path without fixed extreme points. Suppose that $G = (V, E)$, $V = \{v_1, ..., v_n\}$, with edge costs c_e, and distances $d(v_i, v_j)$, is an instance of a path problem. Consider the transformation $G' = (V', E')$ such that $V' = V \cup \{v_0\}$ and $E' = E \cup \{[v_0, v_i] : v_i \in V\}$, with vertex demand, edge costs and distances between vertices defined as follows:

$$w'_i = \begin{cases} w_i, & \text{if } v_i \in V \\ 0, & \text{if } v_i = v_0 \end{cases}$$

$$c'_e = \begin{cases} c_e, & \text{if } e \in E \\ 0, & \text{if } e \in E' \backslash E \end{cases}$$

and

$$d'(v_i, v_j) = \begin{cases} d(v_i, v_j), & \text{if } v_i, v_j \in V \\ 0, & \text{if } v_i = v_j = v_0 \\ \infty, & \text{if } v_i = v_0 \text{ or } v_j = v_0 \end{cases} .$$

With this definition, the cost, covering, and distance sum of a path P in G are the same than those associated to the cycle in G' obtained by linking the extreme points of P to v_0.Then, if $C'_0 = \{v_0, v_{i_1}, ..., v_{i_k}\}$ is an optimal solution for one of the cycle problems with origin v_0 in the graph G', the path $P' = \{v_{i_1}, ..., v_{i_k}\}$ is an optimal solution for the corresponding path problem in G. In fact, if P' were not an optimal solution of the path problem, there would be a better path $P^* = \{v_{j_1}, ..., v_{j_t}\}$ in G. But in that case, the cycle $C^*_0 = \{v_0, v_{j_1}, ..., v_{j_t}\}$ would be a solution better than C'_0 for the cycle problem in G'.

Min-cost s.t. max-cover

Gendreau et al. [25] study the problem of determining the minimum length Hamiltonian cycle in a subset of V, such that all vertices in a set W are within a specified distance r from the cycle. One of the vertices is fixed as depot or start and end point of the tour. This is, using our notation, a *min cost s.t. max cover* formulation with indirect total weighted covering of the demand. An integer linear formulation is given and exact and heuristic solution methods are proposed.

The heuristic method combines the GENIUS heuristic for the TSP (Gendreau et al. [24]) with the PRIMAL1 set covering heuristic of Balas and Ho [4]. The solution is used as an initial upper bound in an exact Branch-and-Cut algorithm.

Both the heuristic and the exact method are tested on randomly generated graphs with up to 600 vertices, of which up to 75 must be in the tour.

Computational results show that the average heuristic gap is smaller than 1% and that the average gap in the root of the Branch-and-Cut is smaller than 0.5%.

2.7 Cycles without a fixed origin

The nine bicriteria problems we consider, involving the location a cycle without a fixed origin on a general graph, are $\mathcal{NP}$-hard, since each of them has the Hamiltonian cycle problem or the TSP as a special case.

Current et al. [17] introduce two bicriteria problems consisting of locating a tour in a graph visiting only n_o vertices. The first objective in both problems is to minimize the total tour length. In one problem, the Maximal Covering Tour Problem (MCTP), the second objective is to maximize the total demand within some prespecified maximal travel distance from a vertex of the tour. In the other problem, the Median Tour Problem (MTP), the second objective consists of minimizig the sum of distances to the vertices not in the facility. So, in the MCTP the objectives are *(minimize cost, maximize cover)*, with indirect covering of the demand, and in the MTP are *(minimize cost, minimize distsum)*. The authors propose integer programming formulations for both problems, and show that the MCTP can be viewed as a particular case of the MTP. Moreover, it is proved that both problems are NP-hard, since the special case in which $n_o = n$ is the TSP. A heuristic procedure for approximating the efficient frontier of the solution set of the MTP is presented.

Min-cost s.t. max-cover

Current et al. [16] have studied the problem with indirect total covering of the demand. This problem, called the Covering Salesman Problem (CSP), is $\mathcal{NP}$-hard since it is a generalization of the TSP. An integer programming formulation is given and a heuristic solution procedure is presented. The heuristic, called COVTOUR, is based upon solution procedures for the set covering problem (SCP) and the TSP. In fact, it consists in solving first a SCP and then solving a TSP on a complete graph induced by the vertex set that is the solution of the SCP. The solution given by COVTOUR is always

feasible, but it may not be optimal and it is not possible to measure its quality. Both the given integer programming formulation and the heuristic can be modified to force certain vertices to be in the solution, and can thus be used to solve the problem of finding the best cycle that visits a given vertex..

Note that the CSP is in some way related with the Generalized Traveling Salesman Problem (GTSP) (Fischetti et al. [19] and [20]), a variant of the TSP in which vertices are partitioned into clusters and the salesman has to visit at least one vertex of each cluster. The sets S_i of vertices that can cover vertex v_i in the CSP can be identified with the clusters of the GTSP.

λ_1(min-cost) + λ_2(min-distsum)

Xu et al. [72] develop a Tabu Search heuristic for a network design problem arising in the telecommunication field. The input elements of the problem are a set of end offices and a set of hubs. Each end office must be connected to exactly one hub, and the selected hubs must be linked by a ring. There are costs associated to selecting a hub, establishing a link between two hubs, and connecting an end office to a hub. The objective is to design such a network at minimum cost, being the function to optimize the sum of the three kind of costs just mentioned.

Tables 2.2 to 2.6 summarize the literature review we present in Sections 2.3 to 2.7. Their aim is to show which problems have been treated in some way and which of them remain unstudied, highlighting in this way the existing gaps. Tables 2.7 to 2.11 summarize the computational complexity results. To describe the covering of the demand in these last five tables we have used a notation of type 'direct (d) - indirect(i) / total(t) - partial(p) / weighted(w) - unweighted(u)'. So, for example, i/t/w means that a weighted graph is considered and that all the demand must be covered indirectly. As to the type of graph on which the problem is considered, we refer by G to general graphs and by T to tree graphs.

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	[14]	[15] [1] [2]	
(oi) s.t. (oj)	[12] [13]		
(oj) s.t. (oi)			

Table 2.2: Literature on paths with fixed end points

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$			
(oi) s.t. (oj)			
(oj) s.t. (oi)		[51] [52] [62]	

Table 2.3: Literature on paths without fixed end points

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	[10] [36] [37] [40] [66]	[40] [41]	
(oi) s.t. (oj)	[37] [43] [49] [42] [18]		
(oj) s.t. (oi)	[67]	[67] [51] [62]	

Table 2.4: Literature on trees

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$			
(oi) s.t. (oj)	[25]		
(oj) s.t. (oi)			

Table 2.5: Literature on cycles with an origin

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$		[72]	
(oi) s.t. (oj)	[16]		
(oj) s.t. (oi)			

Table 2.6: Literature on cycles without a fixed origin

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard
(oi) s.t. (oj)	T : Polynomial [13] G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard
(oj) s.t. (oi)	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard

Table 2.7: Complexity of path with fixed end points problems

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard
(oi) s.t. (oj)	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard
(oj) s.t. (oi)	T : Polynomial G : $\mathcal{NP}$-hard	T : Polynomial G : $\mathcal{NP}$-hard [33] Pseudo-polynomial in series-parallel graphs is partial edges allowed and equality constraint on cost [62]	T : Polynomial G : $\mathcal{NP}$-hard

Table 2.8: Complexity of path without fixed end points problems

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	T : Polynomial [40] [66] G : $\mathcal{NP}$-hard	T : Polynomial [40] [41] G : $\mathcal{NP}$-hard [41]	Polynomial
(oi) s.t. (oj)	$\mathcal{NP}$-hard Polinomial in: - T : i/t/w cover, partial edges allowed [43] - T : d/p/u cover [18] [49]	$\mathcal{NP}$-hard [33]	Polynomial
(oj) s.t. (oi)	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard Polynomial in: - T : partial edges allowed, equality contraint on cost [51] Pseudo-polynomial in: - G : partial edges allowed, equality contraint on cost [62]	Polynomial

Table 2.9: Complexity of tree problems

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard
(oi) s.t. (oj)	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard
(oj) s.t. (oi)	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard

Table 2.10: Complexity of cycles with a fixed origin problems

obj. function	*oi = min-cost* *oj = max-cover*	*oi = min-cost* *oj = min-distsum*	*oi = max-cover* *oj = min-distsum*
$\lambda_1(oi) + \lambda_2(oj)$	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard
(oi) s.t. (oj)	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard
(oj) s.t. (oi)	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard	$\mathcal{NP}$-hard

Table 2.11: Complexity of cycles without a fixed origin problems

Chapter 3

Two Median Cycle Problems

3.1 Introduction and motivation

In Chapter 2 we made an extensive review of the known literature on bicriteria location-routing problems concerning cycles, paths and trees, being the two criteria considered the minimization of the cost and the maximization of the accessibility to the facility. When looking at Table 2.5 we observed that there were several problems still to be explored. This motivated us to study what we have called the *Median Cycle Problem* (MCP), that is defined as follows. Let $G = (V, E \cup A)$ be a complete mixed graph where $V = \{v_1, v_2, \ldots, v_n\}$ is the vertex set, $E = \{[v_i, v_j] \ : \ v_i, v_j \in V, i < j\}$ is the edge set, and $A = \{(v_i, v_j) \ : \ v_i, v_j \in V\}$ is the arc set (loops (v_i, v_i) are included in A). Vertex v_1 is referred to as the *depot*. With each edge $[v_i, v_j]$ is associated a non-negative *routing cost* c_{ij}, and with each arc (v_i, v_j) is associated a non-negative *assignment cost* d_{ij}. A solution to the MCP is a simple cycle through a subset V' of V including v_1 and at least two other vertices. The routing cost of a solution is the sum of the routing costs of all edges on the cycle. The assignment cost of a solution is defined as $\sum_{v_i \in V \setminus V'} \min_{v_j \in V'} d_{ij}$. A solution of the MCP may look like the example in Figure 3.1, which shows an instance with twelve vertices; a tour visits some of them, including the depot v_1, and the others are assigned to visited vertices.

The purpose of this chapter is to provide a model for two versions of the MCP. In the first version, called MCP1, the aim is to determine a solution

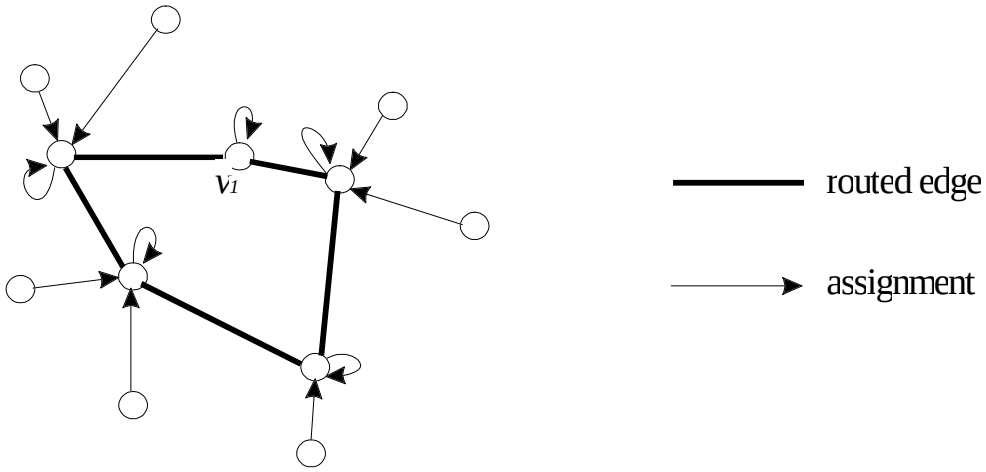

Figure 3.1: A MCP solution

so as to minimize the sum of the routing cost and the assignment cost. In the second version, called MCP2, we seek a solution of least routing cost, subject to an upper bound d_0 on the assignment cost.

3.2 Applications and related problems

A generic application of the MCP is the location of a circular shaped transportation infrastructure such as a metro line or a motorway. Another possible application is the design of a telecommunication network of the type described by Xu et al. [72], in which there are primary nodes or hubs, some of which must be linked by a ring, and secondary nodes that must be assigned to the hubs.

MCP1 consists of minimizing a weighted sum of construction and access costs. In MCP2, only construction costs are considered in the objective function, but a limit is imposed on the assignment cost. Similar problems arise in the design of a collection or delivery routes through a set of locations. Labbé and Laporte [45] consider the case of conveniently locating post-boxes taking into account collection cost and user inconvenience. Another application belonging to the same family is the location of recyclable garbage collection bins in a city.

To the best of our knowledge, the MCP has never previously been studied, but a number of related location-routing problems have been investigated. An important special case of MCP2 is the *Traveling Salesman Problem* (TSP) obtained when $d_0 = 0$. A number of authors have studied the location of a simple cycle on an undirected graph under different objectives or constraints than those of the MCP. We describe briefly some classical examples:

Generalized Traveling Salesman Problem (Fischetti, Salazar and Toth [19] and [20]) : Let us consider an undirected graph with the set of vertices divided in groups or clusters. Each edge has an associated cost. The Generalized Traveling Salesman Problem seeks the minimum cost simple cycle that visits each cluster at least once. Another version of the problem, the Equality Generalized Traveling Salesman Problem, looks for a minimum cost simple cycle that visits each cluster exactly one.

Orienteering Problem (Fischetti, Salazar and Toth [21]; Gendreau, Laporte and Semet [26]): Let us consider an undirected graph where each edge has a cost and each vertex has an associated prize. The Orienteering Problem consists on finding a simple cycle, with bounded cost, that maximizes the total prize collected in the visited vertices.

Prize Collecting Traveling Salesman Problem (Balas [3]): Let us consider an undirected graph where each edge has a cost and each vertex has an associated prize. The Prize Collecting Traveling Salesman Problem seeks a minimum cost cycle that collects at least a given amount on prizes in the visited vertices.

Cycle Problem (Bauer [5]): Let us consider an undirected graph in which each edge has an associated cost. The Cycle Problem calls for finding a minimum cost simple cycle, not necessarily Hamiltonian, in the graph.

Covering Tour Problem (Gendreau, Laporte and Semet [26]): Let us consider an undirected graph with set of vertices $V = U \cup W$. Each edge has an associated cost. The Covering Tour Problem consists of determining a minimum cost simple cycle on a subset of U such that every vertex of W is within a given distance from the cycle.

3.3 Model MCP1

3.3.1 Computational complexity

It is easy to see that the MCP1 is $\mathcal{NP}$-hard, since the TSP is a special case of it. In fact, if for each pair of vertices v_i and v_j the assignment cost is defined as

$$d_{ij} = \begin{cases} \infty, & \text{if } v_i \neq v_j \\ 0, & \text{otherwise} \end{cases},$$

then MCP1 becomes a TSP. However, an even stronger result can be proved.

Proposition 7 *The decision version of MCP1 is $\mathcal{NP}$-complete in the strong sense.*

Proof. We will prove that the decision version of the MCP1 is $\mathcal{NP}$-complete, by showing that there exits a polynomial transformation from the decision Hamiltonian circuit (HC) problem to it.

Decision HC: Given a graph $G = (V, E)$, does G contain a Hamiltonian cycle?

Decision MCP1: Given a complete undirected graph $G = (V, E)$ with routing costs $c_{ij} \geq 0$ associated to all edges $[v_i, v_j] \in E$, a set assignment costs $d_{ij} \geq 0$ associated to all pair of vertices $v_i, v_j \in V$, and a bound $B \in Z^+$, is there a cycle in G such that its length plus the total assignment cost is not larger than B?

The transformation f from HC to MCP1 is defined as follows. Suppose $G = (V, E)$, with $|V| = n$, is an instance of the HC. The corresponding instance $f(G)$ of MCP1 has the same set of vertices. For each pair of vertices v_i and v_j the routing and assignment cost are defined as

$$c_{ij} = \begin{cases} 1 & \text{if } [v_i, v_j] \in E \\ 2 & \text{if } [v_i, v_j] \notin E \end{cases}$$

and

$$d_{ij} = \begin{cases} 0, & \text{if } v_i = v_j \\ 2n, & \text{if } v_i \neq v_j \end{cases},$$

respectively. Moreover, the bound B on the tour length is set to n.

It is easy to see that this transformation f can be performed in polynomial time. For each of the $n(n-1)/2$ costs c_{ij} to specify it is necessary only to examine G to see if the edge $[v_i, v_j]$ exists or not. The value of the n^2 costs d_{ij} is even easier to determine.

We must show now that G contains a Hamiltonian cycle if and only if there is a cycle in $f(G)$ such that the sum of its length and the total assignment cost is not larger than B. First, suppose that there is a Hamiltonian tour in G. Then, the same tour in $f(G)$ verifies that its length plus the total assignment cost is not larger then B, since there are not vertices left outside the tour, and hence the total assignment cost is 0, and the total length of the tour in $f(G)$ is n. Conversely, suppose there is a cycle in $f(G)$ such that its length plus the total assignment cost is not bigger than B. Since $B = n$, no vertex is assigned to a vertex different from itself, that is, the total assignment cost is zero. Moreover its length must be exactly equal to n, which means that all distances between consecutive vertices have value 1 and correspond to edges in G. So the cycle is Hamiltonian in G.

Until now we have proved that the decisional version of MCP1 is $\mathcal{NP}$-complete. Now notice that the largest value that appears in the description of all the instances created by this transformation is $2n$, while the input length is of order n^2 (there are n^2 assignment costs, $n(n-1)/2$ routing cost and a value B). So it is clear that the magnitude of the largest number is bounded by a polynomial of the data length, and even with this restriction the problem is $\mathcal{NP}$-complete. Therefore we can conclude that MCP1 is $\mathcal{NP}$-complete in the strong sense. ■

3.3.2 Mathematical formulation

MCP1 can be formulated as an integer linear program as follows. For each edge $[v_i, v_j] \in E$, let x_{ij} be a binary variable equal to 1 if edge $[v_i, v_j]$ appears on the cycle, and equal to 0 otherwise. For each arc $(v_i, v_j) \in A$, let y_{ij} be a binary variable equal to 1 if vertex v_i is assigned to vertex v_j on the cycle, and equal to 0 otherwise. Notice that if a vertex v_i is on the cycle,

it is then assigned to itself, i.e., $y_{ii} = 1$. In addition, for $S \subset V$ define $E(S) := \{[v_i, v_j] \in E : v_i, v_j \in S\}$ and $\delta(S) := \{[v_i, v_j] \in E : v_i \in S,\ v_j \notin S\}$. If $S = \{v_i\}$, we simply write $\delta(i)$ instead of $\delta(S)$. For $E' \subseteq E$, define $x(E') := \sum_{[v_i,v_j]\in E'} x_{ij}$.

The formulation is then:

$$\text{minimize} \sum_{[v_i,v_j]\in E} c_{ij}x_{ij} + \sum_{(v_i,v_j)\in A} d_{ij}y_{ij} \tag{3.1}$$

subject to:

$$x(\delta(i)) = 2y_{ii} \qquad \text{for all } v_i \in V, \tag{3.2}$$

$$x(\delta(S)) \geq 2y_{ii} \qquad \text{for all } S \subset V : v_1 \notin S, v_i \in S, \tag{3.3}$$

$$\sum_{v_j\in V} y_{ij} = 1 \qquad \text{for all } v_i \in V \setminus \{v_1\}, \tag{3.4}$$

$$y_{ij} \leq y_{jj} \qquad \text{for all } (v_i, v_j) \in A,\ v_i \neq v_j, \tag{3.5}$$

$$y_{11} = 1 \tag{3.6}$$

$$y_{1j} = 0 \qquad \text{for all } v_j \in V \setminus \{v_1\}, \tag{3.7}$$

$$y_{ij} \geq 0 \qquad \text{for all } (v_i, v_j) \in A, \tag{3.8}$$

$$y_{jj} \in \{0, 1\} \qquad \text{for all } v_j \in V \setminus \{v_1\}, \tag{3.9}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for all } [v_i, v_j] \in E. \tag{3.10}$$

Constraints (3.2) are called *degree constraints,* and ensure that the degree of a vertex v_i is 2 if and only if it belongs to the cycle (i.e., $y_{ii} = 1$). Constraints (3.3) are *connectivity constraints* since they state that $S \subseteq V \setminus \{v_1\}$ must be connected to its complement by at least two edges of the cycle whenever at least one vertex $v_i \in S$ is visited by the solution. Constraints (3.4) state that either v_i is a vertex on the cycle (in which case $y_{ii} = 1$), or v_i is assigned to other vertex v_j (in which case $y_{ij} = 1$). We will call them *assignment constraints.* Constraints (3.5) state that a vertex $v_i \in V$ can be assigned to another vertex $v_j \in V$ only if v_j is in the cycle (i.e., $y_{jj} = 1$). Constraints (3.6) and (3.7) ensure that the depot is in the solution. The combination of (3.2), (3.10), (3.6) and (3.7) guarantees that the solution will contain at least one cycle including the depot. Constraints (3.3) limit the number of cycles to one, and the combination of (3.2), (3.4), (3.5), and (3.8) means that every vertex not belonging to the cycle is assigned to a vertex on the cycle. Integrality conditions on the y_{ij} variables with $v_i \neq v_j$ are unnecessary since, for a given integer vector x, the objective is minimized when

vertices not on the cycle are assigned to closest vertices on the cycle. Integer solutions are trivially determined in case of ties. Implicitly the above model imposes that the cycle visits at least three vertices, including the depot. The prove is given below.

Proposition 8 *A feasible solution of the model (3.2)-(3.10) contains a least three vertices, including* v_1.

Proof. Constraint (3.6) states that $y_{11} = 1$. Therefore, the degree constraint (3.2) associated to vertex v_1 becomes

$$x(\delta(1)) = 2.$$

This means that there are two edges incident to vertex v_1, that is, $x_{1i} = x_{1j} = 1$ for some v_i and v_j. Combining these information with the degree constraints for v_i and v_j, we have that

$$2y_{ii} = x(\delta(i)) \geq 1$$

and

$$2y_{jj} = x(\delta(j)) \geq 1.$$

This implies $y_{ii} \geq 0.5$ and $y_{jj} \geq 0.5$. As y_{ii} and y_{jj} are binary variables, the only possibility is $y_{ii} = y_{jj} = 1$. So, the solution contains at least three vertices: v_1, v_i and v_j. ■

3.3.3 Strengthening the LP-relaxation

The linear relaxation of model (MCP1) can be strengthened through the introduction of additional valid inequalities. In this section we present some of these inequalities that we found analizing solutions of the LP-relaxation of model (3.1)-(3.10).

Simple constraints

Clearly, $x_{ij} = 1$ would imply $y_{ii} = 1$ (and also $y_{jj} = 1$). So the classical inequalities

$$x_{ij} \leq y_{ii},\ x_{ij} \leq y_{jj} \text{ for all } v_i, v_j \in V \tag{3.11}$$

are valid for MCP1. Moreover, they are not included in the LP-relaxation of the model (3.2)-(3.10), defined by the constraints (3.2)-(3.8). Consider, for example, an instance with tree vertices and a fractional point (x^*, y^*) with $x^*_{12} = x^*_{13} = 1$, $y^*_{11} = 1$, $y^*_{22} = y^*_{21} = y^*_{33} = y^*_{31} = 0.5$ and all other variables with value 0 (see Figure 3.2). This point satisfies all the constraints in the LP-relaxation, but does not verify $x_{12} \leq y_{22}$ and $x_{13} \leq y_{33}$. Constraints (3.11) will be called *logical constraints*.

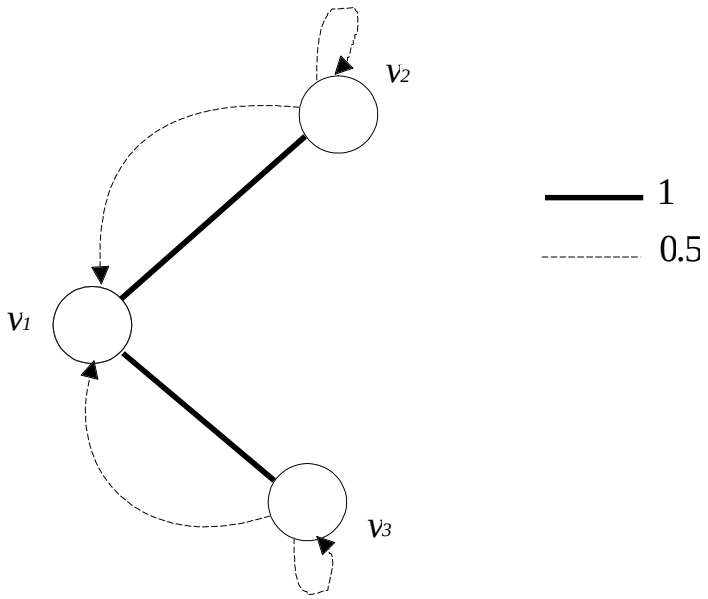

Figure 3.2: Potential solution of the LP-relaxation of MCP1

Proposition 9 *The inequalities*

$$x_{ij} + y_{ki} + y_{kj} \leq y_{ii} + y_{jj} \text{ for all } v_i, v_j, v_k \in V \backslash \{v_1\} \tag{3.12}$$

are valid for MCP1.

Proof. In a feasible solutions for MCP1 the variable x_{ij} has value either 1 either 0. If $x_{ij} = 1$, vertices v_i and v_j are in the solution, that is, $y_{ii} = y_{jj} = 1$, and since $y_{ki} + y_{kj} \leq \sum_{v_j \in V} y_{kj} = 1$, the inequality (3.12) is verified. In the case where $x_{ij} = 0$, we have that

$$x_{ij} + y_{ki} + y_{kj} = y_{ki} + y_{kj} \leq y_{ii} + y_{jj}$$

by using (3.5) for $(v_k, v_i) \in A$ and $(v_k, v_j) \in A$. ■

Figure 3.3 shows a fractional point (x^*, y^*) on an instance with twenty vertices that satisfies all constraints in the LP-relaxation of MCP1 and that, however, violates (3.12). In fact, $x^*_{4,17} + y^*_{19,4} + y^*_{19,17} = 1.57$ which is bigger than $y^*_{4,4} + y^*_{17,17} = 1.14$. We have omitted the representation of variables y_{ii} to simplify the drawing.

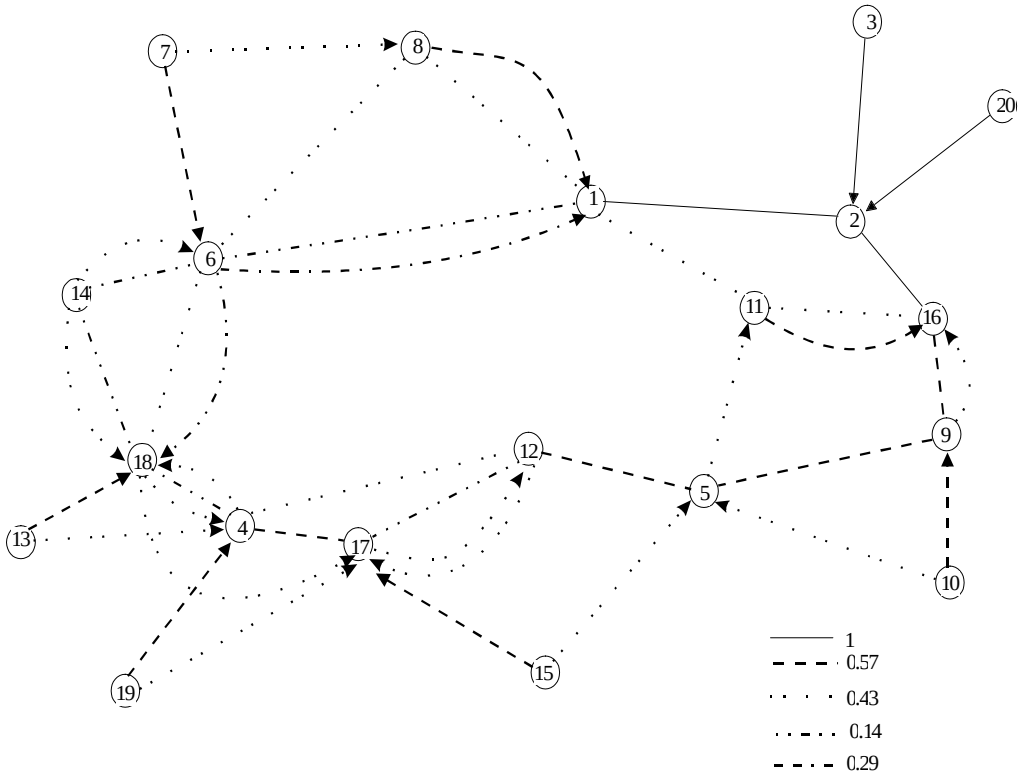

Figure 3.3: Solution of the LP-relaxation of a MPC1 instance

Proposition 10 *The inequalities*

$$x_{ij} + y_{ij} + y_{jk} \leq 1 \text{ for all } v_i, v_j \in V\backslash\{v_1\}, v_k \neq v_j \tag{3.13}$$

are valid for MCP1.

Proof. If edge $[v_i, v_j]$ is in the solution so must be vertices v_i and v_j, that is, $y_{ii} = y_{jj} = 1$, and hence, because of condition (3.4), $y_{ij} = y_{jk} = 0$ (since $v_i \neq v_j$ and $v_k \neq v_j$). If edge $[v_i, v_j]$ is not in the solution then the inequality (3.13) is dominated by inequality (3.5), since $y_{ij} \leq y_{jj} = 1 - \sum_{v_t \in V\backslash\{v_j\}} y_{jt} \leq 1 - y_{jk}$. ■

Inequality (3.13) can be strengthened in the following way:

$$x_{ij} + y_{ij} + \sum_{v_k \in V\backslash\{v_j\}} y_{jk} \leq 1 \text{ for all } v_i, v_j \in V\backslash\{v_1\} \tag{3.14}$$

Figure 3.4 shows a feasible solution for the LP-relaxation of MCP1, on the same instance used in Figure 3.3, that violates (3.13). In fact, $x_{9,10} + y_{10,9} + y_{9,16} = 1.5 > 1$.

Proposition 11 *The inequalities*

$$x_{ij} + y_{ij} \leq y_{jj},\ x_{ij} + y_{ji} \leq y_{ii} \text{ for all } v_i, v_j \in V \tag{3.15}$$

are valid for MCP1.

Proof. If $x_{ij} = 1$, then $y_{ii} = y_{jj} = 1$ and $y_{ij} = y_{ji} = 0$. So, (3.15) is verified. If, on the contrary, $x_{ij} = 0$, then (3.15) is dominated by (3.5). ■

Figure 3.4 also illustrates that this kind of inequalities cannot be derived from the constraints in the LP-relaxation of model (MCP1). In that example the constraint $x_{9,10} + y_{10,9} \leq y_{9,9}$ is violated.

Note that inequalities (3.15) dominate inequalities (3.5) and (3.11). Moreover, they also dominate inequalities (3.14), since

$$x_{ij} + y_{ij} \leq y_{jj} = 1 - \sum_{v_k \in V\backslash\{v_j\}} y_{jk}$$

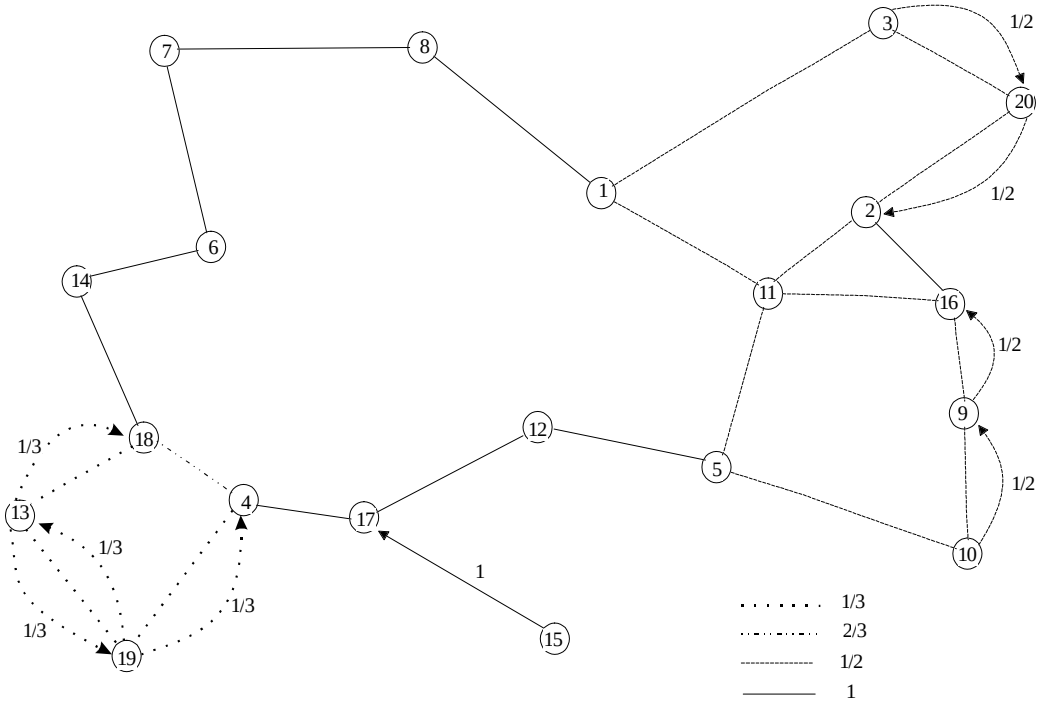

Figure 3.4: Solution of the LP-relaxation of a MCP1 instance

because (3.4), and this implies

$$x_{ij} + y_{ij} + \sum_{v_k \in V \setminus \{v_j\}} y_{jk} \leq 1.$$

Generalized subtour elimination constraints (GSEC)

The nature of our problem allows us to reinforce the subtour elimination constraints (3.3). If there is a set $S \subseteq V \setminus \{v_1\}$ such that a vertex $v_i \in S$ is assigned to another vertex $v_j \in S$, then that vertex v_j is in the solution and so, as $v_1 \notin S$, the set S must be connected to its complement by at least to edges of the solution (see Figure 3.5).

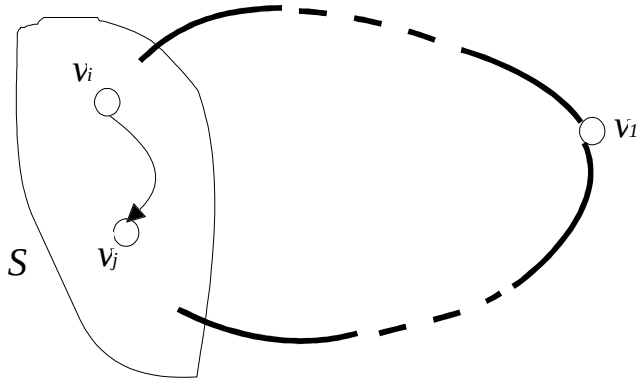

Figure 3.5: GSEC explanation

This is what the following *generalized subtour elimination constraints* state:

$$x(\delta(S)) \geq 2 \sum_{v_j \in S} y_{ij} \text{ for all } S \subseteq V \setminus \{v_1\}, v_i \in S, 2 \leq \mid S \mid \leq \mid V \mid -1 \quad (3.16)$$

First, observe that constraints (3.16) are very rich and reduce to the constraints (3.15) whenever $S = \{i, j\} \subseteq V \setminus \{v_1\}$. Second, constraints (3.16) dominate also constraints (3.12) (just take $S = \{v_i, v_j, v_k\}$). So, these generalized subtour elimination constraints dominate all inequalities in (3.11)-(3.15), except those inequalities (3.15) that refer to v_1, that is

$$x_{1i} \leq y_{ii} \text{ for all } v_i \in V \backslash \{v_1\} \tag{3.17}$$

and

$$x_{1i} + y_{i1} \leq 1 \text{ for all } v_i \in V \backslash \{v_1\}.$$

However, note that the later ones are dominated by the first ones. So, only equations (3.17) remain undominated by (3.16).

The domination relations mentioned until now among these constraints are graphically represented in Figure 3.6.

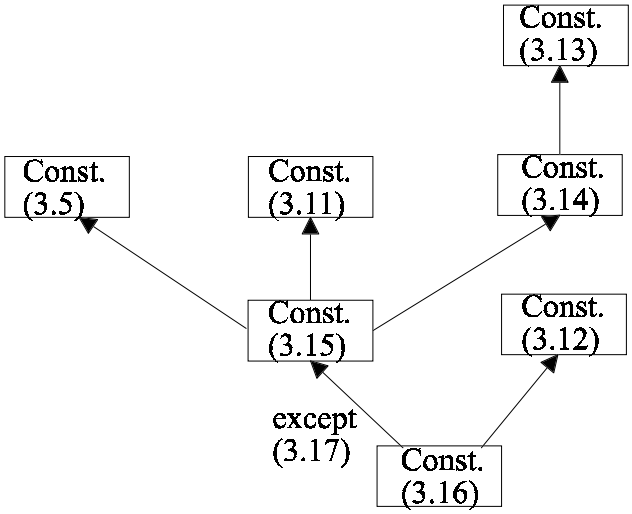

Figure 3.6: Dominance tree

2-matching constraints

Finally, observe that all inequalities valid for the cycle polytope (see Bauer [5]) are also clearly valid for the MCP1. We restrict our attention to the following *2-matching inequalities*:

$$x(E(H)) + x(T) \leq \sum_{v_i \in H} y_{ii} + \frac{|T| - 1}{2} \tag{3.18}$$

for all $H \subset V$ and $T \subset \delta(H)$ satisfying:

(i) $\{v_i, v_j\} \cap \{v_k, v_\ell\} = \emptyset$ for $[v_i, v_j], [v_k, v_\ell] \in T$ and $[v_i, v_j] \not= [v_k, v_\ell]$,

(ii) $|T| \geq 3$ and odd.

These inequalities are obtained by summing up the degree equations (3.2) for all $v_i \in H$ and the bound restrictions $x_{ij} \leq 1$ for all $[v_i, v_j] \in T$, dividing all by two, and rounding down all coefficients to the closest integer.

3.4 Model for MCP2

3.4.1 Computational complexity

The $\mathcal{NP}$-hardness of MPC2 can be easily proved by observing that the special case for which $d_0 = 0$ is the TSP. We present an even stronger result.

Proposition 12 *The decision version of MCP2 is $\mathcal{NP}$-complete in the strong sense.*

Proof. First, we will prove that the decision version of MCP2 is $\mathcal{NP}$-complete by showing that there is a polynomial transformation from the decision HC to it.

Decision MCP2: Given a complete undirected graph $G = (V, E)$ with distances $c_{ij} \geq 0$ for all edges $[v_i, v_j] \in E$, a set of assignment costs $d_{ij} \geq 0$ for all pair of vertices $v_i, v_j \in V$, a value $d_0 \geq 0$, and a bound $B \in Z^+$, does there exist a cycle in G with total assignment cost bounded by d_0 such that its length is not larger than B?

The polynomial transformation f we will use is the one that was defined in the proof of Proposition 7, with the additional condition $d_0 = 0$. We must show now that there is a Hamiltonian tour in G if and only if there is a cycle in $f(G)$ with total assignment cost bounded by d_0 such that the its length is not larger than B. First, suppose that there is a Hamiltonian tour in G. Since there is no vertex outside the tour, this same solution has, in $f(G)$, a total assignment cost equal to zero, and so it is bounded by d_0. Moreover,

its length in $f(G)$ is not larger than B. Conversely, suppose that there is a cycle in $f(G)$ with total assignment cost bounded by d_0 and such that its length is not larger than B. Since d_0 is a finite number, this means that no vertex is assigned to another vertex different from itself, that is, the total assignment cost is zero. On the other hand, the distance between any two consecutive vertices in the tour have to be 1 in order to satisfy the length constraint. This means that $[v_i, v_j] \in E$ for all pair of consecutive vertices in the tour, i.e., the tour is Hamiltonian in G.

To show that the decisional MCP2 is $\mathcal{NP}$-complete in the strong sense it suffices to show that, for each instance created by this transformation, the magnitude of the largest value is bounded by a polynomial of the input data length. This is clear since the largest value that appears in the description of all the instances created by this transformation is $2n$, while the input length is of order n^2 (there are n^2 assignment costs, $n(n-1)/2$ routing cost and a value B). ■

3.4.2 Mathematical formulation

The integer linear programming formulation of MCP2 is identical to that of MCP1 except that the objective becomes

$$\text{minimize} \sum_{[v_i,v_j]\in E} c_{ij}x_{ij} \tag{3.19}$$

and the constraint

$$\sum_{\substack{(v_i,v_j)\in A \\ v_i \neq v_j}} d_{ij}y_{ij} \leq d_0 \tag{3.20}$$

is introduced. Therefore, a valid model for MCP2 is:

$$\text{minimize} \sum_{[v_i,v_j]\in E} c_{ij}x_{ij}$$

subject to

$$\begin{aligned} x(\delta(i)) &= 2y_{ii} \qquad \text{for all } v_i \in V, \\ x(\delta(S)) &\geq 2y_{ii} \qquad \text{for all } S \subset V : v_1 \not\in S, v_i \in S, \end{aligned}$$

$$\begin{aligned}
\sum_{v_j \in V} y_{ij} &= 1 && \text{for all } v_i \in V \setminus \{v_1\}, \\
y_{ij} &\leq y_{jj} && \text{for all } (v_i, v_j) \in A, \\
y_{11} &= 1 && \\
y_{1j} &= 0 && \text{for all } v_j \in V \setminus \{v_1\}, \\
\sum_{\substack{(v_i,v_j)\in A \\ v_i \neq v_j}} d_{ij} y_{ij} &\leq d_0 && \\
y_{ij} &\geq 0 && \text{for all } (v_i, v_j) \in A, \\
y_{jj} &\in \{0,1\} && \text{for all } v_j \in V \setminus \{v_1\}, \\
x_{ij} &\in \{0,1\} && \text{for all } [v_i, v_j] \in E.
\end{aligned}$$

Contraint (3.20) puts an upper bound on the sum of assignment costs from the vertices not in the cycle to vertices in the cycle. Because of it form, we will refer to this constraint as *knapsack constraint*.

3.4.3 Strengthening the LP-relaxation

All the constraints valid for MCP1, such as (3.17), (3.16) and (3.18), are still valid for MCP2. But now, the knapsack constraint (3.20), together with the implicit binary nature of the y_{ij} variables, defines a 0-1 knapsack problem with generalized upper bound (GUB) constraints

$$\sum_{\substack{v_j \in V \\ v_j \neq v_i}} y_{ij} \leq 1$$

for all $v_i \in V$, in which the items are the arcs of the graph. This allow us to reinforce the LP-relaxation of MCP2 by introducing a kind of inequalities that come from the knapsack problem.

Cover inequalities

Let us suppose that there is a set of arcs $T \subseteq A$ with different tails such that the sum of their assignment costs is bigger than d_0. In this case, not

all the variables corresponding to these arcs can have value one in a feasible solution of MCP2. This is what the following *cover inequalities* state:

$$\sum_{(v_i,v_j)\in T} y_{ij} \leq |T| - 1,$$

for all set T of arcs with different tails such that $\sum_{(v_i,v_j)\in T} d_{ij} > d_0$.

Because of the GUB constraints, the cover inequalities can be easily lifted and rewritten as:

$$\sum_{(v_i,v_j)\in \overline{T}} y_{ij} \leq |T| - 1 \tag{3.21}$$

where $\bar{T} := \{(v_i, v_k) : \text{there exists } (v_i, v_j) \in T \text{ for some } v_j \text{ with } d_{ik} \geq d_{ij}\}$ (see Wolsey [70]).

Several strengthenings of these constraints have been suggested (see Nemhauser and Vance [56] and Gu et al. [32]). In particular, the classical extended cover inequality

$$\sum_{(v_i,v_j)\in \overline{T}\cup\widetilde{T}} y_{ij} \leq |T| - 1, \tag{3.22}$$

where $\widetilde{T} := \{(v_i, v_j) \notin \overline{T} : d_{ij} \geq \max_{(k,l)\in\overline{T}}\{d_{kl}\}\}$ are directly relevant in practice to MCP2.

Subtour elimination constraints 2 (SEC2)

Proposition 13 *The SEC2 inequalities*

$$x(\delta(S)) \geq 2 \text{ for all } S \subseteq V\backslash\{v_1\} \text{ such that } \sum_{v_i\in S} \min_{v_j\notin S} d_{ij} > d_0 \tag{3.23}$$

are valid for MCP2.

Proof. If there is a set of vertices $S \subseteq V\backslash\{v_1\}$ such that $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} > d_0$, then at least one vertex of S must be in the tour, which implies that the tour must pass through S. ■

For a set $S \subseteq V\backslash\{v_1\}$ verifying the previous condition, the requirement of being visited by the solution can also be expressed by the inequality

$$\sum_{v_i \in S} y_{ii} \geq 1 \ . \tag{3.24}$$

However, we will use (3.23) because it is stronger. Indeed, for any set $S \subseteq V$

$$\begin{aligned} \sum_{v_i \in S} x(\delta(v_i)) &= 2 \sum_{v_i \in S} y_{ii} \\ &= x(\delta(S)) + 2x(E(S)). \end{aligned}$$

So, $x(\delta(S)) \geq 2$ is equivalent to $\sum_{v_i \in S} y_{ii} \geq 1 + x(E(S))$, which dominates (3.24).

As to the relation between constraints (3.16) and (3.23), note that the maximum value the right hand side of inequalities (3.16) may take is 2. This means that, for a set $S \subseteq V\backslash\{v_1\}$ such that $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} > d_0$, the corresponding constraint (3.23) is stronger than the constraint (3.16).

Cover-connectivity constraints

Consider a set of vertices $S \subseteq V\backslash\{v_1\}$ such that $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} \leq d_0$. Then we cannot impose (3.23). However, if there is an arc $(v_s, v_t) \in (V\backslash S) \times (V\backslash S)$ such that $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} + d_{st} > d_0$ (see Figure 3.7), then it is clear that when $y_{st} = 1$ the solution must visit S in order to verify (3.20).

We can thus conclude that, for such a set S of vertices and such an arc (v_s, v_t),

$$x(\delta(S)) \geq 2y_{st}.$$

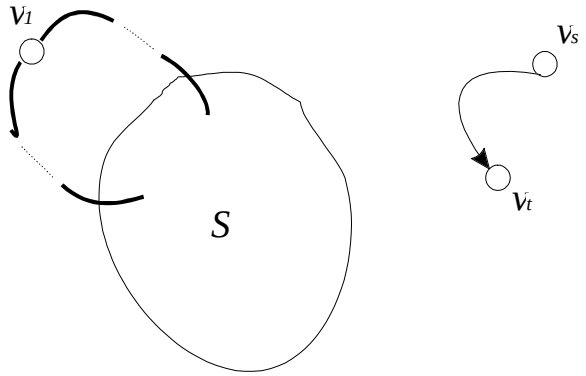

Figure 3.7: Cover-connectivity explanation

This idea can be extended to a bigger set of arcs. For example, suppose that $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} + d_{st}$ is smaller or equal than d_0, but there is another arc $(v_p, v_q) \in (V \backslash S) \times (V \backslash S)$, with $v_p \neq v_s$, and $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} + d_{st} + d_{pq} > d_0$. Then any solution in which $y_{st} + y_{pq} = 1$ must visit S, that is

$$x(\delta(S)) \geq 2(y_{st} + y_{pq} - 1).$$

In general, we obtain the following type of constraints that will be called *cover-connectivity constraints*:

$$x(\delta(S)) \geq 2\Big(\sum_{(v_i,v_j) \in T} y_{ij} - (|T| - 1) \Big), \tag{3.25}$$

where S is a set of vertices not containing v_1 and T is a set of arcs with different tails in $V \backslash S$ such that

$$\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} + \sum_{(v_s,v_t) \in T} d_{st} > d_0.$$

Moreover, (3.25) can be reinforced by adding to T all those arcs (v_s, v_k) such that there exists a vertex v_t for which $(v_s, v_t) \in T$ and $d_{sk} \geq d_{st}$ or $v_k \in S$.

The following proposition presents these ideas in a more formal way.

Proposition 14 *The cover-connectivity constraints*

$$x(\delta(S)) \geq 2\Big(1 - |T| + \sum_{(v_i,v_j)\in\bar{T}} y_{ij}\Big) \tag{3.26}$$

for all $S, T, \bar{T}$ *such that*

(i) $S \subset V \setminus \{v_1\}$,

(ii) T *is a subset of arcs* (v_i, v_j) *with different tails belonging to* $V \setminus S$,

(iii) $\bar{T} := \{(v_i, v_k) :$ *there exists* $(v_i, v_j) \in T$ *for some* v_j *with* $d_{ik} \geq d_{ij}$ *or* $v_k \in S\}$,

are valid inequalities for MCP2 model whenever

$$\sum_{v_i\in S} \min_{v_j\notin S}\{d_{ij}\} + \sum_{(v_i,v_j)\in T} d_{ij} > d_0. \tag{3.27}$$

Proof. Let $W = \{v_i \in V : (v_i, v_j) \in T\}$. First observe that $\sum_{(v_i,v_j)\in\overline{T}} y_{ij} \leq |T|$ since $|W| = |T|$ and constraints (3.4) apply. If $\sum_{(v_i,v_j)\in\overline{T}} y_{ij} \leq |T| - 1$, constraint (3.26) holds trivially. If $\sum_{v_i,v_j\in\overline{T}} y_{ij} = |T|$, constraint (3.26) reduces to $x(\delta(S)) \geq 2$. In fact, two cases are then possible:

(i) There exists $(v_i, v_k) \in \overline{T}$ with $v_k \in S$ and $y_{ik} = 1$. Then at least one vertex v_k of S must be on the cycle since v_i is assigned to it and $x(\delta(S)) \geq 2$ must hold.

(ii) All $(v_i, v_k) \in \overline{T}$ such that $y_{ik} = 1$ satisfies $v_i \notin S$ and $v_k \notin S$. Then the left-hand side of (3.27) is a lower bound on the assignment cost if no vertex of S belongs to the cycle, so that again $x(\delta(S)) \geq 2$ must hold. ■

After having presented the cover-connectivity constraints some remarks are in order, to highlight their richness. First, observe that when $T = \emptyset$, constraints (3.26) reduce to

$$x(\delta(S)) \geq 2$$

for all $S \subseteq V \backslash \{v_1\}$ such that $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} > d_0$, that is, they reduce to (3.23). Second, note that when $S = \emptyset$, the constraints can be written as

$$\sum_{(v_i,v_j) \in \overline{T}} y_{ij} \leq |T| - 1$$

under the condition $\sum_{(v_i,v_j) \in T} d_{ij} > d_0$. These are nothing else that the cover constraints (3.21) we presented before. So the cover-connectivity constraints generalize both (3.21) and (3.23). These domination relations are graphically represented in Figure 3.8.

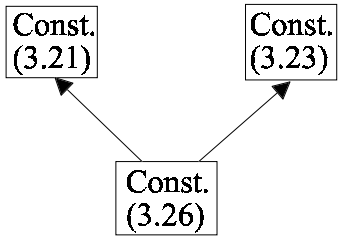

Figure 3.8: Dominance tree

Chapter 4

Polyhedral study of MCP1

In this chapter we derive some polyhedral results for model (3.1)–(3.10) defining MCP1. In Sections 4.1 and 4.2 we will assume that $|V| \geq 6$ in order to eliminate trivial or degenerate instances, and to be able to apply known results for the TSP polytope when required. In Section we study the MCP1 polytope when $|V| = 3$, $|V| = 4$, and $|V| = 5$.

Let P be the polytope defined by the convex hull of feasible solutions to MCP1, that is,

$$P := \text{conv}\left\{(x, y) \in \mathbb{R}^{|E|+|A|} : (x, y) \text{ satisfies (3.2)–(3.10)} \right\}.$$

Let Q be the polytope associated with the TSP. Note that Q can be described as

$$Q := \{(x, y) \in P : y_{ii} = 1 \text{ for all } v_i \in V \setminus \{v_1\}\}.$$

In fact, both polytopes P and Q are linked by the intermediate polytopes:

$$P(F) := \{(x, y) \in P : y_{ii} = 1 \text{ for all } v_i \notin F \cup \{v_1\}\},$$

for all $F \subseteq V \setminus \{v_1\}$, in the sense that $P(\emptyset) = Q$ and $P(V \setminus \{v_1\}) = P$. The vertices in set F are called said to be *free*, since they are not compulsorily in the solution.

This connection allows us to extend known results from Q to P. For any $\alpha \in \mathbb{R}^{|E|}, \beta \in \mathbb{R}^{|A|}$ and $\gamma \in \mathbb{R}$, define the hyperplane

$$\mathcal{H}(\alpha, \beta, \gamma) := \{(x, y) \in \mathbb{R}^{|E|+|A|} : \alpha x + \beta y = \gamma\}.$$

Lemma 15 *Let $v_{i_1}, v_{i_2}, ..., v_{i_{n-1}}$ be an ordering of the vertices of $V \setminus \{v_1\}$ and let $F_k = \{v_{i_1}, ..., v_{i_k}\}$ for all $k = 1, \ldots, n-1$. If for each $k = 1, ..., n-1$ and each $v_{i_t} \in V \setminus \{v_{i_k}\}$ there exists a feasible solution (x, y) to (MCP1) such that*

(i) $y_{i_l i_l} = 1$ for all $l > k$ (i.e., all vertices of $V \setminus F_k$ belong to the cycle),

(ii) $y_{i_l i_l} + y_{i_l 1} = 1$ for all $l < k$ (i.e., all vertices of $F_k \setminus \{v_{i_k}\}$ are either on the cycle or assigned to the depot v_1),

(iii) $y_{i_k i_t} = 1$ (i.e., vertex v_{i_k} does not belong to the cycle and is assigned to vertex v_{i_t}), and

(iv) $\alpha x + \beta y = \gamma$,

then $\dim(\mathcal{H}(\alpha, \beta, \gamma) \cap P) \geq \dim(\mathcal{H}(\alpha, \beta, \gamma) \cap Q) + (|V| - 1)^2$.

Proof. We will prove by induction on $|F_k|$ that $\dim(\mathcal{H}(\alpha, \beta, \gamma) \cap P(F_k)) \geq \dim(\mathcal{H}(\alpha, \beta, \gamma) \cap Q) + |F_k|(|V| - 1)$. If $|F_k| = 0$, i.e., if $F_k = \emptyset$, then this inequality holds since $P(F_k) = Q$. Now, by the induction hypothesis, there exist $\dim(\mathcal{H}(\alpha, \beta, \gamma) \cap Q) + |F_{k-1}|(|V| - 1) + 1 = \dim(\mathcal{H}(\alpha, \beta, \gamma) \cap Q) + |F_k - 1|(|V| - 1) + 1$ affinely independent points such that $y_{i_k i_k} = 1$ in $\mathcal{H}(\alpha, \beta, \gamma) \cap P(F_{k-1})$ and thus in $\mathcal{H}(\alpha, \beta, \gamma) \cap P(F_k)$. An additional set of $|V| - 1$ affinely independent in $\mathcal{H}(\alpha, \beta, \gamma) \cap P(F_k)$ can be obtained as follows. Take the points consisting of a Hamiltonian cycle in $V \setminus \{v_{i_k}\}$ and assign v_{i_k} to each vertex in the cycle in turn. Note that in all these points $y_{i_k i_k} = 0$. Writing the points (x, y) as row vectors with variables in the order $(edges, nodes, arcs)$ we obtain the matrix in Figure 4.1.

	edges	*nodes*	*arcs*
		$y_{i_k i_k}$	$y_{i_k 1}$ $\cdots$ $y_{i_k n}$
(a)		1 $\vdots$ 1	0 $\cdots$ 0 $\vdots$ $\vdots$ 0 $\cdots$ 0
(b)		$1 \cdots 0 \cdots 1$ $\vdots \quad \vdots \quad \vdots$ $1 \cdots 0 \cdots 1$	1 $\ddots$ 1

Figure 4.1: Affinely independent points in $\mathcal{H}(\alpha, \beta, \gamma) \cap P(F_k)$

Vectors in part (a) of the matrix represent the $\dim(\mathcal{H}(\alpha,\beta,\gamma)\cap Q)+|F_{k-1}|(|V|-1)+1$ points existing due to the induction hypothesis, and vectors in part (b) represent the new $|V|-1$ points. It is easily seen that all the rows of this matrix are linearly independent. ■

An important consequence of Lemma 15 is that any facet-defining inequality $\alpha x+\beta y\leq\gamma$ for Q satisfying (i)–(iv) is also facet-defining for P. In the forthcoming sections we will use this fact to present alternative and simpler proofs (referred in what follows as *Lemma-based proofs*) for the dimension and facet defining results.

4.1 Dimension and trivial facets

We start by studying the dimension of the intermediate polytopes $P(F)$ defined in the previous section.

Proposition 16 $\dim(P(F))\geq|E|-|V|+|F|(|V|-1)$.

Proof. The proof is by induction on $|F|$. If $F=\emptyset$ then $P(F)=Q$ and the inequality holds since $\dim(Q)=|E|-|V|$. Let us suppose that the inequality holds for a given $F'\neq\emptyset$, and consider the set $F''=F'\cup\{v_k\}$, $v_k\in V\backslash\{F'\cup\{v_1\}\}$. We must prove that

$$\begin{aligned}\dim(P(F'')) &\geq |E|-|V|+|F''|(|V|-1)\\ &= |E|-|V|+(|F'|+1)(|V|-1)\\ &= |E|-|V|+|F'|(|V|-1)+(|V|-1)\end{aligned}$$

By the induction hypothesis, there exist at least $|E|-|V|+|F'|(|V|-1)+1$ affinely independent vectors in $P(F')$, and thus in $P(F'')$. The additional $|V|-1$ affinely independent points we need can be obtained in the following way: take a Hamiltonian tour in $V\backslash\{v_k\}$ and assign v_k to each vertex of the tour, in turn. The affine independence of the vectors can be seen in the matrix in Figure 4.2: vectors (a) are the $|E|-|V|+|F'|(|V|-1)+1$ affinely independent vectors in $P(F')\subseteq P(F'')$), vectors (b) are the ones associated to the $|V|-1$ new solutions. It can be seen that the $|V|-1$ new vectors are mutually affinely independent. They are also affinely independent with

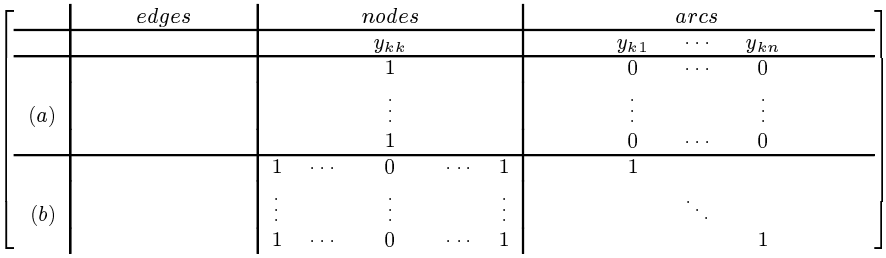

Figure 4.2: Affinely independent points in $P(F'')$

all vectors in (a) since in all those the variable y_{kk} has value 1, while it has value 0 in the all vectors in (b).

Therefore, $\dim(P(F'')) \geq |E| - |V| + |F'|(|V| - 1) + (|V| - 1) = |E| - |V| + |F''|(|V| - 1)$. ■

Proposition 17 $\dim(P(F)) = |E| - |V| + |F|(|V| - 1)$.

Proof. Given Proposition 16 it only remains to show that $\dim(P(F)) \leq |E| - |V| + |F|(|V| - 1)$. This follows from the linear programming formulation associated to $P(F)$, which is:

$$
\begin{aligned}
x(\delta(i)) = 2y_{ii} \qquad & \text{for all } v_i \in V, \\
x(\delta(S)) \geq 2y_{ii} \qquad & \text{for all } S \subset V : v_1 \not\in S, v_i \in S, \\
\sum_{v_j \in V} y_{ij} = 1 \qquad & \text{for all } v_i \in F, \\
y_{ij} \leq y_{jj} \qquad & \text{for all } (v_i, v_j) \in A, \\
y_{ii} = 1 \qquad & \text{for all } v_i \in V \backslash F \\
y_{ij} = 0 \qquad & \text{for all } v_i \in V \backslash F, v_j \in V \setminus \{v_1\}, \\
y_{ij} \geq 0 \qquad & \text{for all } (v_i, v_j) \in A, \\
y_{jj} \in \{0, 1\} \qquad & \text{for all } v_j \in V \setminus \{v_1\}, \\
x_{ij} \in \{0, 1\} \qquad & \text{for all } [v_i, v_j] \in E.
\end{aligned}
$$

In this model there are, for each vertex $v_i \notin F$, $|V|$ equations ($y_{ii} = 1$, and $y_{ij} = 0$ for all $v_j \neq v_i$), and therefore the corresponding equations (3.4)

are redundant. So, in the model there are $|V|^2+|V|+\mid F \mid (|V|-1)$ linearly independent equations ($|V|$ of type (3.2), $|F|$ of type (3.4), and $(|V|-|F|)|V|$ of type $y_{ii}=1$ and $y_{ij}=0$). As the number of variables is $|E|+|V|^2$, it follows that $\dim(P(F)) \leq |E|-|V|+\mid F \mid (|V|-1)$. ■

We can now determine easily the dimension of the MCP1 polytope.

Proposition 18 $\dim(P) = |E|+|V|^2-3|V|+1$.

Proof.

Direct proof:

This result is a corollary of Proposition 17 obtained when taking $F = V\backslash\{v_1\}$.

Lemma-based proof:

Recall that we are assuming $|V| \geq 6$. Then

$$\begin{aligned}
\dim(P) &= \dim(\mathcal{H}(0,0,0)\cap P) \\
&\geq \dim(\mathcal{H}(0,0,0)\cap Q)+(|V|-1)^2 \quad \text{(by Lemma 15 using any sequence)} \\
&= \dim(Q)+(|V|-1)^2 \\
&= |E|-|V|+(|V|-1)^2 \quad \text{(see Grötschel and Padberg [31]).}
\end{aligned}$$

On the other hand, $\dim(P) \leq |E|+|V|^2-3|V|+1$ since (MCP1) can be described with $|E|+|V|^2$ variables and $|V|$ type (3.2), $|V|-1$ type (3.4), and $|V|$ type (3.6)–(3.7) linearly independent equality constraints. ■

We present now some facet defining results.

Proposition 19 *The inequality $x_{ij} \geq 0$ defines a facet of P for all $[v_i, v_j] \in E$.*

Proof.

Direct proof:

The proof is by induction on $|F|$. If $F=\emptyset$ then $P(F)=Q$ and the statement is true (Grötschel and Padberg [31]). Let us suppose it holds for a given $F \neq \emptyset$, and consider the set $F' = F \cup \{v_k\}$, $v_k \in V\backslash\{F \cup \{v_1\}\}$. By the induction hypothesis, $x_{ij} \geq 0$ defines a facet of $P(F)$, that is, there

are $\dim(P(F))+1$ affinely independent vectors in $P(F)$, and thus in $P(F')$, satisfying $x_{ij}=0$. Since $\dim(P(F'))=\dim(P(F))+(|V|-1)$, to prove that $x_{ij}\geq 0$ induces a facet of $P(F')$ we must find $|V|-1$ new affinely independent vectors in $P(F')$ satisfying $x_{ij}=0$. These points can be obtained by taking a Hamiltonian tour in $V\backslash\{v_k\}$ such that edge $[v_i,v_j]$ is not in the tour, and by assigning v_k successively to every vertex in the tour. Writing all these points (x,y) as rows with variables in the order *(edges, nodes, arcs)*, we obtain the matrix in Figure 4.3.

	edges	*nodes*	*arcs*
	x_{ij}	y_{kk}	y_{k1} $\cdots$ y_{kn}
(a)	0	1	0 $\cdots$ 0
	$\vdots$	$\vdots$	$\vdots$ $\quad$ $\vdots$
	0	1	0 $\cdots$ 0
(b)	0	1 $\cdots$ 0 $\cdots$ 1	1
	$\vdots$	$\vdots$ $\quad$ $\vdots$	$\ddots$
	0	1 $\cdots$ 0 $\cdots$ 1	1

Figure 4.3: Affinely independent points satisfying $x_{ij}=0$

The vectors in (a) correspond to the $\dim(P(F))+1$ affinely independent points in $P(F')$, and the vector in (b) to the new $|V|-1$ points. As can be easily deduced from the matrix, all these vectors are affinely independent.

Lemma-based proof:

The hyperplane defined by $x_{ij}=0$ satisfies the assumptions of Lemma 15 for any sequence. Indeed for any vertex $v_k\in V\backslash\{v_1\}$, a simple cycle spanning $V\backslash\{v_k\}$ and not containing edge $[v_i,v_j]$, together with the assignment of v_k to any vertex of the cycle, yields an (x,y) vector of P such that $x_{ij}=0$. Hence, $\dim(\{(x,y)\in P: x_{ij}=0\})\geq\dim(\{(x,y)\in Q: x_{ij}=0\})+(|V|-1)^2=(|E|-|V|-1)+(|V|-1)^2$, since $x_{ij}\geq 0$ induces a facet of Q (see Grötschel and Padberg [31]). The thesis then follows for Proposition 19. ■

Note at this stage that the inequalities $x_{ij}\leq 1$ are not facet inducing since they are dominated by the constraints (3.11), since $x_{ij}\leq y_{jj}\leq 1$.

Proposition 20 *The inequality $y_{ij}\geq 0$ defines a facet of P for all $(v_i,v_j)\in A$, $i\neq j$ and $i\neq 1$.*

Proof.

Direct proof:

We will show that there are $\dim(P) + 1$ affinely independent vectors in P that satisfy $y_{ij} = 0$.

a) Since $Q \subset P$, there are $|E| - |V| + 1$ affinely independent vectors in P corresponding to Hamiltonian tours in V, and having thus $y_{ij} = 0$.

b) For each $v_k \in V \backslash \{v_1, v_i\}$, let us consider the $|V| - 1$ points consisting of a Hamiltonian tour in $V \backslash \{v_k\}$ with v_k assigned to a vertex in the tour. This makes a total of $(|V| - 2)(|V| - 1) = |V|^2 - 3|V| + 2$ vectors that are affinely independent among them and with the vectors in (a).

c) The remaining $|V| - 2$ vectors can be obtained by taking a Hamiltonian tour in $V \backslash \{v_i\}$ (this is always possible because $v_i \neq v_1$) and by successively assigning v_i to every vertex in the tour except v_j.

These three kind of points are graphically represented by the rows of the matrix in Figure 4.4. It is easily seen that all rows are linearly independent.

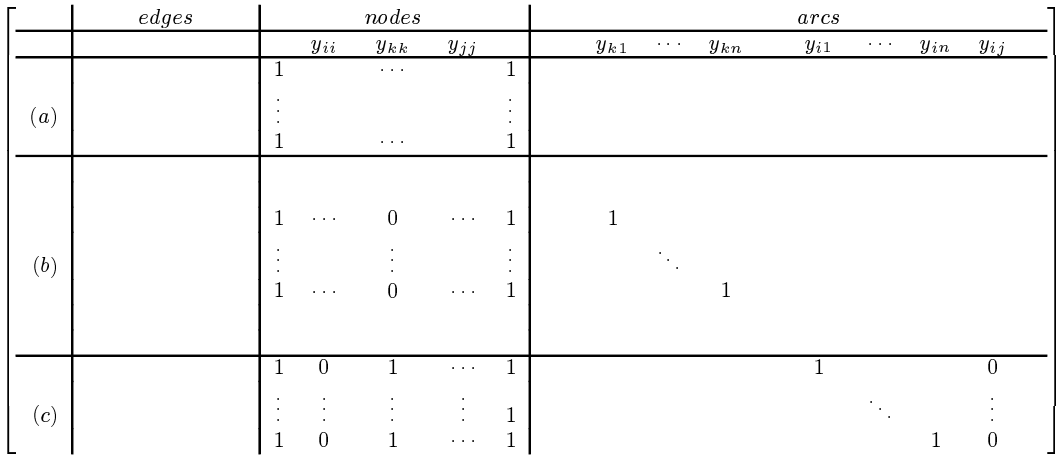

Figure 4.4: Affinely independent points satisfying $y_{ij} = 0$

Lemma-based proof:

The proof is obtained along the lines of the proof of Lemma 15 except that F is now initialized as $F = \{v_i\}$. Since in this case $Q \subset P(F)$, there exist

$\dim(Q)+1 = |E|-|V|+1$ affinely independent vectors (Hamiltonian cycles) such that $y_{ij} = 0$. An additional $|V|-2$ affinely independent vectors are obtained by taking a simple cycle spanning $V \setminus \{v_i\}$ and assigning v_i to each vertex of the cycle different from v_j (so $y_{ij} = 0$), which proves that $y_{ij} \geq 0$ defines a facet of $P(F)$ for $F = \{v_i\}$. Moreover, for all $v_k \in V \setminus \{v_1, v_i\}$, any simple cycle spanning $V \setminus \{v_k\}$ and the successive assignment of v_k to the vertices of the cycle yield (x, y) vectors of P satisfying conditions (i)–(iv) of Lemma 15. ■

If $v_i = v_j$, the inequality $y_{ij} \geq 0$ is not facet inducing since it is implied by the combination of equalities (3.2) and $x_{ij} \geq 0$ for $[v_i, v_j] \in E$. In fact, because of contraints (3.2),

$$y_{ii} = x(\delta(i))/2 \geq 0.$$

The inequality $y_{ij} \leq 1$ is not facet inducing since it is implied by the combination of constraints (3.4) and (3.8). In fact, because of constraints (3.8) and (3.4),

$$y_{ij} \leq \sum_{v_k \in V} y_{ik} = 1.$$

4.2 Other facets

In this section we prove that some of the inequalities presented in Section 3.3.3 are not only valid but also facet defining for the MCP1 polytope.

Proposition 21 *The valid inequality $x(\delta(S)) \geq 2\sum_{v_j \in S} y_{ij}$ defines a facet of P for all $S \subseteq V \setminus \{v_1\}$, $2 \leq |S| \leq |V|-3$, $v_i \in S$.*

Proof.

Direct proof:

The proof is by induction on $|F|$. If $F = \emptyset$ then $P(F) = Q$ and (3.16) reduces to $x(\delta(S)) \geq 2$, which is a facet of Q (Grötschel and Padberg [31]). So, let us assume that $x(\delta(S)) \geq 2\sum_{v_j \in S} y_{ij}$ defines a facet of $P(F)$ for a

given set $F \subseteq V\backslash\{v_1\}$, $F \neq \emptyset$, and consider the set $F' = F \cup \{v_k\}$, $v_k \in V\backslash\{F \cup \{v_1\}\}$. We must prove that $x(\delta(S)) \geq 2\sum_{v_j \in S} y_{ij}$ also defines a facet of $P(F')$. We will show that there are $\dim(P(F')) + 1$ affinely independent vectors in $P(F')$ that satisfy (3.16) at equality.

Note that $\dim(P(F')) = \dim P(F) + (|V| - 1)$. By the induction hypothesis we know that there are $\dim(P(F)) + 1$ affinely independent vector in $P(F)$, and so in $P(F')$, that satisfy (3.16) at equality. All these points have $y_{kk} = 1$. We need another $|V|-1$ affinely independent points. We distinguish two cases:

i) $v_k \neq v_i$. The $|V|-1$ new solutions are built up by taking a Hamiltonian tour in $V \setminus \{v_k\}$ and by assigning v_k to each of the vertices in the tour, one after the other. All these points verify $x(\delta(S)) = 2\sum_{v_j \in S} y_{ij} = 2$.

Representing the points as row vectors we obtain the matrix in Figure 4.5.

	edges	*nodes*	*arcs*
		y_{kk}	y_{k1} $\cdots$ y_{kn}
(c)		1 $\vdots$ 1	0 $\cdots$ 0 $\vdots$ $\quad$ $\vdots$ 0 $\cdots$ 0
(a)		$1 \cdots 0 \cdots 1$ $\vdots \quad \vdots \quad \vdots$ $1 \cdots 0 \cdots 1$	1 $\ddots$ 1

Figure 4.5: Affinely independent points satisfying (3.16) at equality

Rows (c) represent the vectors given by the induction hypothesis, and rows (a) the new built ones. It is clear that all vectors are affinely independent.

ii) $v_k = v_i$. If v_i is set free now, it means that the previous $\dim(P(F))+1$ affinely independent vectors had $y_{ii} = 1$. By taking a Hamiltonian tour in $V \setminus \{v_i\}$ and assigning v_i to each of the vertices $v_j \in S\backslash\{v_i\}$ we obtain $|S| - 1$ new linearly independent solutions satisfying $x(\delta(S)) = 2\sum_{v_j \in S} y_{ij} = 2$. Another $|V| - |S|$ solution can be obtained taking a Hamiltonian tour in $V\backslash S$ (this is always possible because $|V| \geq 6$), and assigning v_i to each of the vertices in the tour. These former tours verify $x(\delta(S)) = 2\sum_{v_j \in S} y_{ij} = 0$.

The two kind of solutions in (ii) sum up a total of $|V|-1$ points which are clearly linearly independent among them and with the $\dim(P(F))+1$ first points, as shows the matrix representation in Figure 4.6. Rows (c) represent the vectors given by the induction hypothesis, and rows (b1) and (b2) correspond, respectively, to the $|S|-1$ and $|V|-|S|$ new points.

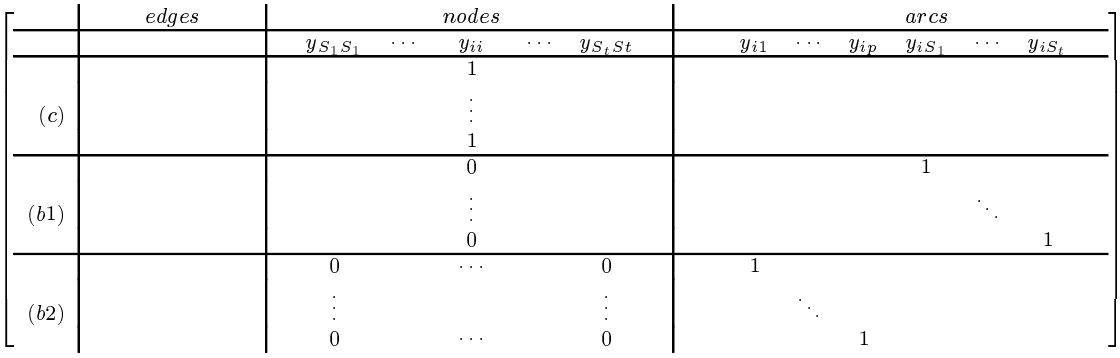

Figure 4.6: Affinely independent points satisfying (3.16) at equality

Lemma-based proof:

The hyperplane defined by $x(\delta(S)) = 2\sum_{v_j \in S} y_{ij}$ satisfies the assumptions of Lemma 15 for any sequence containing v_i in last position. Indeed, for any vertex $v_k \in V \setminus \{v_1, v_i\}$ there exists a simple cycle spanning $V \setminus \{v_k\}$ and with two edges in $\delta(S)$ such that, together with the assignment of v_k to each vertex of the cycle, yields (x, y) vectors of P satisfying $x(\delta(S)) = 2\sum_{v_j \in S} y_{ij} (\equiv 2)$. Furthermore, for v_i, the last vertex of the sequence, a simple cycle spanning $S \cup \{v_1\} \setminus \{v_i\}$ with two edges in $\delta(S)$ can be constructed. The assignment of v_i to each vertex of $S \setminus \{v_i\}$ yields $|S|-1$ solutions (x, y) satisfying the hypothesis (i) to (iv) of Lemma 15. An additional $|V \setminus S|$ such solutions (x, y) are obtained by constructing a simple cycle spanning $V \setminus S$ (which contains at least three vertices), and successively assigning v_i to each vertex of $V \setminus S$. The thesis follows by noting that in Q, the inequality $x(\delta(S)) \geq 2\sum_{v_j \in S} y_{ij}$ reduces to $x(\delta(S)) \geq 2$, which is facet defining (Grötschel and Padberg, [31]). ■

Proposition 21 does not hold if $|S| = 1$ or $|S| = |V| - 2$ since constraints (3.16) are then dominated by (3.2) or $x(\delta(S)) \geq 2$, respectively. The validity of the latter inequality comes from the fact that the cycle must con-

tain at least one vertex from S whenever $S \subset V \setminus \{v_1\}$ and $|S| \geq |V| - 2$. Proposition 21 is also invalid if $v_i \notin S$ since then

$$x(\delta(S \cup \{v_i\})) = x(\delta(S)) + x(\delta(i)) - 2 \sum_{v_j \in S} x_{ij} \geq 2 \sum_{v_j \in S \cup \{v_i\}} y_{ij} = 2(y_{ii} + \sum_{v_j \in S} y_{ij}).$$

This implies

$$x(\delta(S)) \geq 2 \sum_{v_j \in S} (y_{ij} + x_{ij}),$$

which dominates the corresponding constraint (3.16).

Proposition 22 *The inequality $x_{1j} \leq y_{jj}$ defines a facet of P for $j \neq 1$.*

Proof.

Direct proof:

The proof is by induction on $|F|$. If $F = \emptyset$ then $P(F) = Q$ and $x_{1j} \leq y_{jj}$ reduces to $x_{1j} \leq 1$, which is a facet of Q (Grötschel and Padberg [31]). Let us suppose that Proposition 22 holds for a given set $F \subseteq V \setminus \{v_1\}$, $F \neq \emptyset$, and consider the set $F' = F \cup \{v_k\}$, $v_k \in V \setminus (F \cup \{v_1\})$. We must prove that $x_{1j} \leq y_{jj}$ defines a facet of $P(F')$, that is, that there are $\dim(P(F')) + 1$ affinely independent points in $P(F')$ satisfying $x_{1j} = y_{jj}$. Because of the induction hypothesis we know that there are $\dim(P(F)) + 1$ points in $P(F)$, and so in $P(F')$, with these properties. As $\dim(P(F')) = \dim P(F) + (|V| - 1)$, it remains to find another $|V| - 1$ points. We distinguish two cases

i) $v_k = v_j$. If v_j is set free now it means that in the first $\dim(P(F)) + 1$ vectors the variable y_{jj} had value 1. The new $|V| - 1$ points are obtained by taking a Hamiltonian tour in $V \setminus \{v_j\}$ and assigning v_j to each of the vertices in the tour. All these solutions would verify $x_{1j} = y_{jj} = 0$. If we represent the points as row vectors we obtain the matrix in Figure 4.7.

Vectors (c) are those given by de induction hypothesis, and vectors (a) are the new ones. It is clear that all row vectors are affinely independent.

ii) $v_k \neq v_j$. The new $|V| - 1$ points are obtained by taking a Hamiltonian tour in $V \setminus \{v_k\}$ that includes edge $[v_1, v_j]$, and assigning v_k to each of the vertices in the tour. All these solutions would verify $x_{1j} = y_{jj} = 1$. Again, representing the points as row vectors we obtain the matrix in Figure 4.8.

	edges	*nodes*					*arcs*		
	x_{1j}	y_{jj}					y_{j1}	$\cdots$	y_{jn}
		1					0	$\cdots$	0
		$\vdots$					$\vdots$		$\vdots$
(c)		1					0	$\cdots$	0
	0	1	$\cdots$	0	$\cdots$	1	1		
	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\ddots$	
(a)	0	1	$\cdots$	0	$\cdots$	1			1

Figure 4.7: Affinely independent points satisfying $x_{1j} = y_{jj}$

	edges	*nodes*					*arcs*		
	x_{1j}	y_{kk}					y_{k1}	$\cdots$	y_{kn}
		1					0	$\cdots$	0
		$\vdots$					$\vdots$		$\vdots$
(c)		1					0	$\cdots$	0
	1	1	$\cdots$	0	$\cdots$	1	1		
	$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\ddots$	
(b)	1	1	$\cdots$	0	$\cdots$	1			1

Figure 4.8: Affinely independent points satisfying $x_{1j} = y_{jj}$

Vectors (c) are those given by the induction hypothesis, and vectors (b) are the new ones. It is clear that all row vectors are affinely independent.

Lemma-based proof:

The hyperplane defined by $x_{1j} = y_{jj}$ satisfies the assumptions of Lemma 15. This can be seen by taking a simple cycle spanning $V \setminus \{v_k\}$ and successively assigning v_k to each vertex of the cycle. Furthermore, this cycle must contain edge $[v_1, v_j]$ whenever $v_k \neq v_j$. The thesis follows from the fact that $x_{1j} \leq y_{jj} = 1$ induces a facet of Q (Grötschel and Padberg [31]). ■

The constraints $x_{i1} + y_{i1} \leq 1$ for all $v_i \neq v_1$ do not induce facets of P since they are dominated by the constraints $x_{1i} \leq y_{ii}$. Indeed,

$$x_{1i} \leq y_{ii} = 1 - \sum_{v_j \in V \setminus \{v_i\}} y_{ij} \leq 1 - y_{i1}.$$

Proposition 23 *The inequality* $x(E(H)) + x(T) \leq \sum_{v_i \in H} y_{ii} + (|T| - 1)/2$ *for all* $H \subset V$ *and* $T \subset \delta(H)$ *satisfying:*

(i) $\{v_i, v_j\} \cap \{v_k, v_\ell\} = \emptyset$ *for* $[v_i, v_j], [v_k, v_\ell] \in T$ *and* $[v_i, v_j] \neq [v_k, v_\ell]$,

(ii) $|T| \geq 3$ *and odd,*

defines a facet of P.

Proof.

Direct proof:

We use induction on $|F|$. If $F = \emptyset$ then $P(F) = Q$ and $y_{ii} = 1$ for all $v_i \in V$, so the two matching inequality (3.18) reduces to

$$x(E(H)) + x(T) \leq \mid H \mid + \frac{|T| - 1}{2},$$

which defines a facet of Q.

Let us suppose that (3.18) defines a facet of $P(F)$, for a given $F \subseteq V \backslash \{v_1\}$, $F \neq \emptyset$, and let us consider the set $F' = F \cup \{v_i\}$, $v_i \in V \backslash \{F \cup \{v_1\}\}$. To prove that (3.18) also defines a facet of $P(F')$ we must find $\dim(P(F')) + 1 = \dim(P(F)) + (|V| - 1) + 1$ affinely independent point in $P(F')$ satisfying (3.18) at equality. By the induction hypothesis we know that $\dim(P(F)) + 1$ such points exist, all of them having $y_{ii} = 1$; it remain to find another $|V| - 1$.

We distinguish two cases:

i) $v_i \notin H$. We take a Hamiltonian tour in $V \backslash \{v_i\}$ such that $x_{st} = 1$ for all $[v_s, v_t] \in T$ except for one (the one incident in v_i, if there is any), and $x_{st} = 1$ for exactly $|H| - (|T| - 1)/2$ edges in $E(H)$. Observe that for such a solution (3.18) is satisfied at equality, since

$$\begin{aligned} x(E(H)) + x(T) &= \left(|H| - \frac{|T| - 1}{2} \right) + (|T| - 1) \\ &= |H| + \frac{|T| - 1}{2} \\ &= y(H) + \frac{|T| - 1}{2}. \end{aligned}$$

ii) $v_i \in H$. We take a Hamiltonian tour in $V\backslash\{v_i\}$ such that $x_{st} = 1$ for all $[v_s, v_t] \in T$ except for one (the one incident in v_i, if there is any), and $x_{st} = 1$ for exactly $(|H|-1)-(|T|-1)/2$ edges in $E(H)$. Again, such solution satisfies (3.18) is satisfied at equality, since

$$\begin{aligned} x(E(H)) + x(T) &= \left((|H|-1) - \frac{|T|-1}{2}\right) + (|T|-1) \\ &= (|H|-1) + \frac{|T|-1}{2} \\ &= y(H) + \frac{|T|-1}{2}. \end{aligned}$$

Therefore, in both cases, (i) and (ii), it is possible to find a Hamiltonian tour in $V\backslash\{v_i\}$ satisfying (3.18) at equality. The $|V|-1$ affinely independent solutions we need are obtained by assigning v_i to each of the nodes in that cycle.

Lemma-based proof:

The hyperplane defined by $x(E(H)) + x(T) \leq \sum_{v_i \in H} y_{ii} + (|T|-1)/2$ satisfies the assumptions of Lemma 15. Indeed, for each vertex $v_k \in V \setminus \{v_1\}$ a simple cycle spanning $V \setminus \{v_k\}$ and satisfying hypothesis (i)–(iv) of Lemma 15 can be designed as described in the direct proof, and v_k can be assigned to each vertex of the cycle. The thesis follows from the fact that $x(E(H)) + x(T) \leq \sum_{v_i \in H} y_{ii} + (|T|-1)/2$ defines a facet of Q (Grötschel and Padberg [31]). ■

4.3 Small MCP1 polytopes

Up to now in this chapter we have assumed that there are at least 6 vertices in the graph. This was done in order to eliminate degenerated cases and also to be able to apply known results for the TSP polytope when required.

In this section we give a complete linear description of the MCP1 polytope P for the sizes of V that remain unstudied, that is, $|V| = 3$, $|V| = 4$ and $|V| = 5$. These descriptions were generated using the software package PORTA, designed by Th. Christof, M. Jünger and G. Reinelt (see Th. Christof [9])

4.3.1 MCP1 polytope when $|V| = 3$

When $|V| = 3$, P contains only one feasible point (the only possible solution is a cycle of three edges), whose coordinates are shown in Table 4.1. Therefore, $\dim(P) = 0$.

edges			*nodes*			*arcs*					
1	1	1	1	1	1	0	0	0	0	0	0

Table 4.1: Only feasible point for MCP1 with $|V| = 3$

4.3.2 MCP1 polytope when $|V| = 4$

When $|V| = 4$, P is the convex hull of the 12 points, that correspond to 3 cycles of four edges and 3 cycles with three edges visiting the depot, combined with all possible assignments. Their coordinates are shwon in Table 4.2.

edges						*nodes*				*arcs*											
1	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	0	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0
1	0	1	0	1	0	1	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	1	1	0	0	1	1	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0

Table 4.2: Feasible points for MCP1 with $|V| = 4$

Note that all 12 points are affinely independent. Therefore, $\dim(P) = 11$.

A complete and nonredundant characterization of P is given by the following set of equalities and inequalities:

$$\begin{aligned}
x(\delta(i)) = 2y_{ii} \quad & \text{for all } v_i \in V, \\
\sum_{v_j \in V} y_{ij} = 1 \quad & \text{for all } v_i \in V \setminus \{v_1\}, \\
y_{11} = 1 \quad & \\
y_{1j} = 0 \quad & \text{for all } v_j \in V \setminus \{v_1\}, \\
x_{1j} \leq y_{jj} \quad & \text{for all } v_j \in V \setminus \{v_1\}, \\
y_{2j} \geq 0 \quad & \text{for all } v_j \in V \setminus \{v_2\}, \\
y_{3j} \geq 0 \quad & \text{for all } v_j \in V \setminus \{v_3\}, \\
y_{4j} \geq 0 \quad & \text{for all } v_j \in V \setminus \{v_4\}.
\end{aligned}$$

The number of distinct facets is 12.

4.3.3 MCP1 polytope when $|V| = 5$

When $|V| = 5$, P is the convex hull of the 114 points, that represent all feasible solutions (12 cycles of five edges, 12 cycles of four edges visiting the depot, and 6 cycles of three edges visiting the depot, all of them combined with all possible assignments in each case).

A complete and nonredundant characterization of P is given by the following set of equalities and inequalities:

$$\begin{aligned}
x(\delta(i)) = 2y_{ii}, \quad & \text{for all } v_i \in V, \\
y_{11} = 1, \quad & \\
y_{1j} = 0 \quad & \text{for all } v_j \in V \setminus \{v_1\} \\
\sum_{v_j \in V} y_{ij} = 1 \quad & \text{for all } v_i \in V \setminus \{v_1\}, \\
x_{1j} \geq 0 \quad & \text{for all } v_j \in V \setminus \{v_1\}, \\
x_{1j} \leq y_{jj}, \quad & \text{for all } v_j \in V \setminus \{v_1\},
\end{aligned}$$

$$
\begin{aligned}
\left.\begin{array}{r} x_{ij} + y_{ij} \leq y_{jj} \\ x_{ij} + y_{ji} \leq y_{ii} \end{array}\right\} \quad & \text{for all } [v_i, v_j] \in E, v_i \neq v_1, \\
y_{2j} \geq 0, \quad & \text{for all } v_j \in V \setminus \{v_2\}, \\
y_{3j} \geq 0, \quad & \text{for all } v_j \in V \setminus \{v_3\}, \\
y_{4j} \geq 0, \quad & \text{for all } v_j \in V \setminus \{v_4\}, \\
y_{5j} \geq 0, \quad & \text{for all } v_j \in V \setminus \{v_5\}, \\
x_{ij} \geq 0, \quad & \text{for all } [v_i, v_j] \in E, v_i \neq v_1 \\
y_{35} + y_{54} + y_{43} \leq 1, & \\
y_{53} + y_{34} + y_{45} \leq 1, & \\
y_{42} + y_{25} + y_{54} \leq 1, & \\
y_{24} + y_{45} + y_{52} \leq 1, & \\
y_{25} + y_{53} + y_{32} \leq 1, & \\
y_{52} + y_{23} + y_{35} \leq 1, & \\
y_{43} + y_{32} + y_{24} \leq 1, & \\
y_{34} + y_{42} + y_{23} \leq 1. &
\end{aligned}
$$

The number of distinct facets is 50.

Since all the fourteen equalities are linearly independent and the number of variables involved is 35 ($|E| = 10$ and $|A| = 25$), the dimension of the polytope is at most 21. On the other hand, the 22 feasible points shown in Table 4.3, correspoding to four cycles with three edges and six cycles with five edges, are affinely independent. Therefore, $\dim(P) = 21$.

edges										*nodes*														*ar*
0	1	0	1	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	1	0	0	0
0	1	0	1	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	1	1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	1	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0

Table 4.3: 22 affinely independent feasible points for MCP1 with $|V| = 5$

Chapter 5

Branch-and-Cut algorithms

In this chapter we describe the main features of a Branch-and-Cut algorithm for the two models of MCP. In Section 5.1 we study the separation problems for the valid inequalities described in the two previous chapters. Section 5.2 outlines the Branch-and-Cut procedure.

5.1 Separation procedures

In this section we outline the procedures used to separate the different types of cuts used within our Branch-and-Cut algorithm. We denote by $G^* = (V^*, E^*)$ the *support graph* associated with a given (fractional) solution (x^*, y^*), i.e., $V^* := \{v_i \in V : y^*_{ii} > 0\}$ and $E^* := \{[v_i, v_j] \in E : x^*_{ij} > 0\}$. We also define $A^* := \{(i, j) \in A : y^*_{ij} > 0\}$.

5.1.1 Generalized subtour elimination constraints

The GSEC inequalities (3.16) involving sets S such that $|S| = 2$ reduce to the inequalities (3.15), that is, $x_{ij} + y_{ij} \leq y_{jj}$, and any of them can be violated only if the corresponding edge $[v_i, v_j]$ belongs to E^* or the corresponding arc (v_i, v_j) belongs to A^*. So, they are separated by examining only edges in E^* and arcs in A^*. The exhaustive search of violated inequalities (3.15) can be done in $O(|V|^2)$.

Let us consider now the separation problem for general GSEC. If the support graph G^* is not connected, then each subset $S \subset V$ corresponding to a connected component not containing vertex v_1 and each $v_i \in S$ generate a violated generalized subtour elimination constraint. If G^* is connected, finding a most violated generalized subtour elimination constraint

$$x(\delta(S)) \geq 2 \sum_{v_j \in S} y_{ij}$$

is equivalent to finding the largest violation of

$$x(\delta(S)) + 2 \sum_{v_j \notin S} y_{ij} \geq 2 \sum_{v_j \in S} y_{ij} + 2 \sum_{v_j \notin S} y_{ij} = 2 \sum_{v_j \in V} y_{ij} = 2.$$

For a given vertex $v_i \in V \setminus \{v_1\}$ this reduces to a maximum flow problem defined as follows. Let $G^{**} = (V^{**}, E^{**})$ be a graph such that $V^{**} = V^* \cup \{v_{n+1}\}$ and $E^{**} = E^* \cup \{[v_j, v_{n+1}] : v_j \in V^*\}$. The capacity of each edge $e \in E^*$ is equal to x^*_e and that of the new edges $[v_j, v_{n+1}]$ is equal to $2y^*_{ij}$ if $v_j \neq v_i$, and to infinity if $v_j = v_i$ (see Figure 5.1).

Let $S' \subset V^{**}$ be such that $v_{n+1} \in S'$, $v_1 \notin S'$ and the capacity Δ of the cut $\delta(S')$ in G^{**} is minimum. If $\Delta \geq 2$, the current solution (x^*, y^*) does not violate any generalized subtour elimination constraint involving v_i. Otherwise, $S = S' \backslash \{v_{n+1}\}$ defines a most violated generalized subtour elimination constraint (3.16) involving v_i. Note that, because of the huge capacity associated to edge $[v_i, v_{n+1}]$, vertex v_i will always belong to S. Algorithm 1 describes this exact separation procedure.

Finding the maximum flow between two vertices in a graph with n vertices requires $O(n^3)$ operations, so the overall complexity of Algorithm 1 is $O(|V|^4)$.

5.1.2 2-matching inequalities

The 2-matching inequalities can be separated in polynomial time using a modification of the odd cuts separation procedure proposed by Padberg and

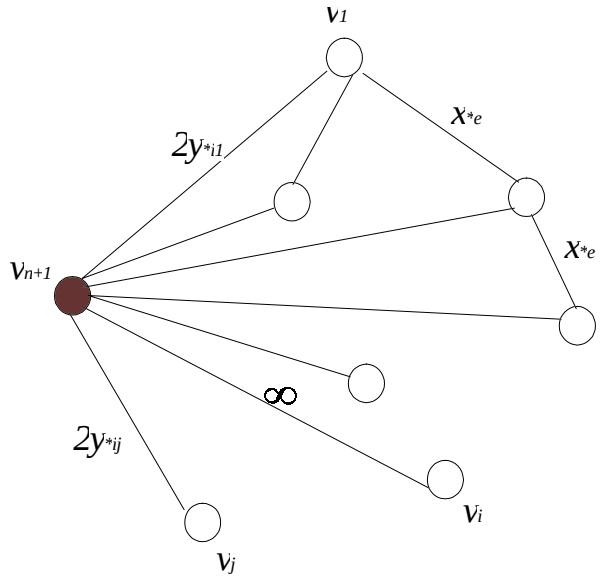

Figure 5.1: Graph G^{**}

Rao [58]. However, to reduce the computational time we have implemented the heuristic given by Fischetti, Salazar and Toth [21], which is described in Algorithm 2.

Algorithm 2 is based on the same idea that the greedy Kruskal's algorithm for finding the Minimum Spanning Tree of a graph. The edges in E^* are sorted by increasing value of x^*_e. At each iteration the algorithm selects a new edge $e \in E^*$, providing that it does not make a cycle when added to the previously selected ones. Let H be the connected component of the subgraph made up by the selected edges that contains e. The set of vertices H is then considered as the handle in a possible 2-matching constraint. The teeth are determined in the following way. Let $\delta(H) = \{e_1, ..., e_p\}$, with $x^*_{e_1} \geq ... \geq x^*_{e_p}$. Initially we relax the requirement of empty intersection between all pairs of teeth. For each odd value $|T|$ between 3 and p, the best possible choice for the teeth are $e_1, ..., e_{|T|}$. So, an inequality with maximum violation would correspond to the number $|T|$ that maximizes $x(T) - (|T| - 1)/2$. If proceeding this way no violated inequality is found, then no violated 2-matching inequality exits for the current handle H. Otherwise, the set of vertices H and the set of edges T verify $x(E(H)) + x(T) \leq \sum_{v_i \in H} y_{ii} + (|T| - 1)/2$, there might be some pairs of teeth incident to the same vertex. Let us

Algorithm 1 Separation of GSEC

Data: the support graph $G^* = (V^*, E^*)$

if G^* is not connected **then**
  each connected component S such that $v_1 \notin S$ and each $v_i \notin S$ generate a violated GSEC
else
  for $v_i \in V^* \backslash \{v_1\}$ **do**
    create G^{**}
    find the minumun cut S' between v_{n+1} and v_i, and let Δ be its value
    if $\Delta < 2$ **then**
      $S' \backslash \{v_{n+1}\}$ and v_i define a violated GSEC
    end if
  end for
end if

suppose that e and f are two such teeth, both incident to vertex v. Then we define a new couple (H', T') of handle and teeth as $T' = T \backslash \{e, f\}$ and $H' = H \backslash \{v\}$, if $v \in H$, or $H' = H \cup \{v\}$ if $v \notin H$. The new 2-matching inequality associated to (H', T') is at least as violated as the one associated to (H, T). In fact, replacing (H, T) by (H', T') the increment in the violation is equal to $1 - y_{vv} \geq 0$, if $v \in H$, or is at least $1 + y_{vv} - x(\delta(v)) \geq 2y_{vv} - x(\delta(v)) = 0$, if $v \notin H$. This step is repeated until no pair of teeth intersects. If at the end of the process $|T| = 1$, being $T = \{[v_i, v_j]\}$, with $v_j \notin H$, then the 2-matching inequality is discarded since it is dominated by the generalized subtour elimination constraint associated to v_j and the cut $H \cup \{v_j\}$.

5.1.3 Cover-connectivity inequalities

In order to separate constraints such as generalized subtour elimination constraints (3.16) or 2-matching constraints (3.18), we generate subsets $S \subset V \backslash \{v_1\}$. If $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} > d_0$ and $x^*(\delta(S)) < 2$ for a given subset S, then the cover-connectivity constraint (3.27) corresponding to S and $T = \emptyset$ reduces to a SEC2 (3.23) that is violated by the current solution (x^*, y^*). The SEC2 is then added to the current LP-relaxation, preferably to the constraint we are trying to separate while building S.

Now consider the separation problem for the general case. Specifically,

Algorithm 2 Sepration of 2-matching inequalites

Data: the support graph $G^* = (V^*, E^*)$

sort the edges in E^* by increasing order of x_e^*
$Tree := \emptyset$
for all $e \in E^*$ **do**
  if e does not make a cycle when added to $Tree$ **then**
    $H := \{$connected componet of $Tree$ that contains $e\}$
    sort the edges in $\delta(H)$ by decreasing order of x_e^*, so that $\delta(H) := \{e_1, ..., e_p\}$
    choose the odd value $|T|$ between 3 and p that maximizes $x(T) - (|T| - 1)$
    $T := \{x_{e_1}, ..., x_{e_{|T|}}\}$
    while there are edges in T with common vertices **do**
      take a pair of edges f and g in T with a common vertex v
      $T := T \backslash \{f, g\}$
      if $v \in H$ **then**
        $H := H \backslash \{v\}$
      else
        $H := H \cup \{v\}$
      end if
    end while
    if $|T| = 1$ **then**
      the vertex of the tooth that does not belong to H, and H plus that same vertex define a violated GSEC
    else
      (H, T) defines a violated 2-matching inequality
    end if
  end if
end for

the problem amounts to determining sets $S \subset V\backslash\{v_1\}$ and $T \subset A$ such that:

(i) if $(v_i, v_j) \in T$ then $v_i \in V\backslash S$ and $v_j \in V\backslash S$,

(ii) no two different arcs of T have the same tail,

(iii) $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} + \sum_{(v_i,v_j)\in T} d_{ij} > d_0$, and

(iv) $x^*(\delta(S)) + \sum_{(v_i,v_j)\in T} 2(1 - w^*_{ij}) < 2$, where $w^*_{ij} := \sum\{y^*_{ik} : d_{ik} \geq d_{ij}$ or $v_k \in S\}$.

Condition (iv) means that the cover-connectivity inequality associated with S and T is violated by the current solution (x^*, y^*), since $\sum_{(v_i,v_j)\in T} 2(1 - w^*_{ij}) = 2(|T| - \sum_{(v_i,v_j)\in \overline{T}} y^*_{ij})$. Observe that even if the set S (respectively T) is fixed, finding an appropriate set T (respectively S) is not a classical knapsack (respectively, minimum cut) problem. In fact the separation problem for the cover-connectivity constraints is $\mathcal{NP}$-hard. To show it we will prove that even the separation of the special case of cover-connectivity constraints resulting from considering $T = \emptyset$ (that is, SEC2) is $\mathcal{NP}$-hard.

Proposition 24 *The separation problem for SEC2 (3.23) is $\mathcal{NP}$-hard.*

Proof. Given a fractional solution (x^*, y^*), the separation problem for constraints SEC2, consists on determining whether there is a subset $S \subset V\backslash\{v_1\}$ with $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} > d_0$ for which $x(\delta(S)) < 2$.

Suppose we are given the graph $G = (V, E)$ such that $E = \{[v_1, v_i] : i = 2, ..., n\}$ (see Figure 5.2) and $d_{ij} = 0$ if $v_i \neq v_1$ and $v_j \neq v_1$.

In such graph, the separation problem for constraints (3.23) can be formulated as

$$\sigma^* = \min \sum_{v_i \in V\backslash\{v_1\}} x^*_{1i} z_i$$

subject to

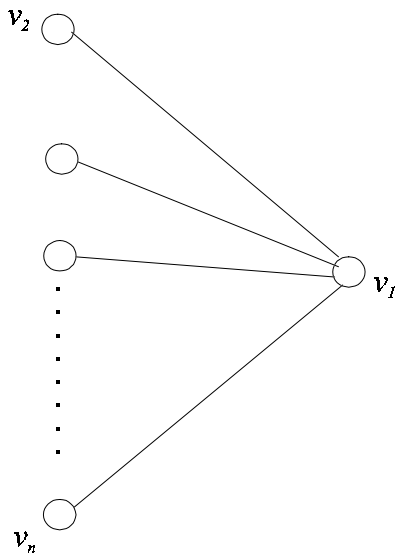

Figure 5.2: Example graph

$$\sum_{v_i \in V\setminus\{v_1\}} d_{i1} z_i \geq d_0 + 1$$
$$z_i \in \{0,1\} \qquad \text{for all } v_i \in V\setminus\{v_1\},$$

and this is a 0-1 knapsack problem, which is known to be $\mathcal{NP}$-hard. ■

We therefore use a heuristic procedure for separating cover-connectivity constraints in which sets S and T are built sequentially. More precisely, we combine the following two polynomial greedy procedures.

Heuristic 1: Greedy procedure for T

Let (x^*, y^*) be a fractional solution and let S be a given subset of $V\setminus\{v_1\}$ such that $x^*(\delta(S)) < 2$ and $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} \leq d_0$ (if $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} > d_0$ then we would have a violated SEC2. The heuristic looks for a set T of arcs in $V\setminus S$ and with different tails, such that

$$\sum_{(v_i,v_j)\in T} 2(1 - w^*_{ij}) < 2 - \underbrace{x^*(\delta(S))}_{\text{fixed}}$$

and

$$\sum_{(v_i,v_j)\in T} d_{ij} > d_0 - \underbrace{\sum_{v_i\in S} \min_{v_j\notin S} d_{ij}}_{\text{fixed}} .$$

It proceeds as follows. For each vertex $v_i \notin S$, let $v_{j(i)}$ be such that $j(i) := \arg\max\{w^*_{ij} : v_j \notin S\}$, set

$$score(v_i) := \frac{2(1 - w^*_{ij(i)})}{d_{ij(i)}},$$

and rank the vertices $v_i \in V \setminus S$ in non-decreasing order of their score. (If $d_{ij(i)} = 0$ then $score(v_i)$ is set to an arbitrary large number.) Pairs $(v_i, v_{j(i)})$ are iteratively selected according to their ranking until condition (iii) is satisfied. This provides a first candidate arc set for T. Then, the arc that was selected last is removed and subsequent arcs in the list are added until (iii) holds again. This step is repeated until all vertices in the list have been considered. Every candidate set for T satisfying (iv) obtained in this way also define, together with S, a cover-connectivity constraints violated by the current fractional solution (x^*, y^*).

Algorithm 3 gives a pseudo-code description of the procedure.

Heuristic 2: Greedy procedure for S

Let (x^*, y^*) be a fractional solution and let T be a given subset of arcs satisfying condition (ii) and such that $\sum_{(v_i,v_j)\in T} 2(1-w^*_{ij}) < 2$. If $\sum_{(v_i,v_j)\in T} d_{ij} > d_0$, then T and $S := \emptyset$ define a violated cover inequality (3.21). In such case, we add to the current LP-relaxation the classical extended version of cover inequality

$$\sum_{(v_i,v_j)\in \overline{T}\cup\bar{T}} y_{ij} \le |T| - 1,$$

where $\bar{T} := \{(v_i, v_k) :$ there exists $(v_i, v_j) \in T$ for some v_j with $d_{ik} \ge d_{ij}$ or $v_k \in S\}$ and $\tilde{T} := \{(v_i, v_j) \notin \overline{T} : d_{ij} \ge \max_{(k,l)\in T}\{d_{kl}\}\}$.

If T satisfies $\sum_{(v_i,v_j)\in T} d_{ij} < d_0$, we use a sequential procedure to determine a suitable set $S \subset V\setminus(V(T) \cup \{v_1\})$, where $V(T)$ denotes the set of all vertices incident to arcs of T, such that

Algorithm 3 Separation of cover-conncetivity constraints. Greedy 1

Data: the support graph $G^* = (V^*, E^*)$, A^*, and a set $S \subset V \backslash \{v_1\}$ such that $x(\delta(S)) < 2$ and $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} < d_0$

$T := \emptyset$
for all $v_i \notin S$ **do**
  for all $v_j \notin S, v_j \neq v_i$ **do**
    $w^*_{ij} := \sum\{y^*_{ik} : d_{ik} \geq d_{ij}$ or $v_k \in S\}$
  end for
  $j(i) := \arg\max\{w^*_{ij} : v_j \notin S\}$
  if $d_{ij(i)} \neq 0$ **then**
    $score(v_i) := 2(1 - w^*_{ij(i)})/d_{ij(i)}$
  else
    $score(v_i) := \infty$
  end if
end for
Let $L := \{v_{i_1}, ..., v_{i_p}\}$ be the list of vertices $v_i \notin S$ sorted by increasing order of their score
$k := 1$
while $k \leq p$ **do**
  $T := T \cup \{(v_{i_k}, v_{j(i_k)})\}$
  if $\sum_{v_i \in S} \min_{v_j \notin S} d_{ij} + \sum_{(v_i,v_j) \in T} d_{ij} > d_0$ **then**
    if $x^*(\delta(S)) + \sum_{(v_i,v_j) \in T} 2(1 - w^*_{ij}) < 2$ **then**
      S and T define a violated cover-connectiviy constraint
      $T := T \backslash \{(v_{i_k}, v_{j(i_k)})\}$
    else
      return
    end if
  end if
  $k := k + 1$
end while

$$x^*(\delta(S)) < 2 - \underbrace{\sum_{(v_i,v_j)\in T} 2(1-w^*_{ij})}_{\text{fixed}}$$

and

$$\sum_{v_i\in S} \min_{v_j\notin S} d_{ij} > d_0 - \underbrace{\sum_{(v_i,v_j)\in T} d_{ij}}_{\text{fixed}} .$$

We start with $S = \emptyset$. Then, at each iteration, all vertices v_j in $V\backslash(V(T)\cup\{v_1\}\cup S)$ are given the following labels:

$$\begin{aligned} l_1(v_j \mid S) &:= x^*\left(\delta(S\cup\{v_j\})\right) - x^*(\delta(S)) + 2 \\ &= 2\left(y^*_{jj} - x^*\left(\delta(j)\cap\delta(S)\right) + 1\right) \end{aligned}$$

and

$$l_2(v_j \mid S) := \sum_{v_i\in S\cup\{v_j\}} \min_{v_k\notin S\cup\{v_j\}} d_{ik} - \sum_{v_i\in S} \min_{v_k\notin S} d_{ik}.$$

Label $l_1(v_j \mid S)$ measures the contribution of v_j to $x^*(\delta(S))$, and thus to the left-hand side of (iv) if v_j is included in S. The constant 2 is added to ensure the nonnegativity of the label. Similarly, label $l_2(v_j \mid S)$ captures the contribution of v_j to the left-hand side of (iii), and it is positive by definition.

Each $v_j \in V\backslash(V(T)\cup\{v_1\}\cup S)$ is then given a score and the vertex with the smallest score is added to S. The procedure stops when condition (iii) is fulfilled, and the cover-connectivity constraint associated with S and T is then added to the current LP-relaxation if condition (iv) also holds.

Given a set S, the score of vertex $v_j \in V\backslash(V(T)\cup\{v_1\}\cup S)$ is defined as the ratio of the two labels:

$$score(v_j \mid S) := \frac{l_1(v_j \mid S)}{l_2(v_j \mid S)}.$$

At each iteration and for all $v_j \in V$ we save

$$nearest(v_j \mid S) := \arg\min\{d_{jk} : v_k \notin S\cup\{v_j\}\}.$$

Every time a vertex, say v_i, is included in S, the labels of all vertices $v_j \notin S \cup V(T) \cup \{v_1\}$ are updated as follows:

$$l_1(v_j \mid S \cup \{v_i\}) = l_1(v_j \mid S) - 2x^*_{ij},$$

and $l_2(v_j \mid S \cup \{v_i\})$ is computed from $l_2(v_j \mid S)$ by using $nearest(v_j \mid S)$.

Algorithm 4 gives a pseudo-code description of the procedure.

Algorithm 4 Separation of cover-conncetivity constraints. Greedy 2

Data: the support graph $G^* = (V^*, E^*)$, A^*, and a set T of arcs with diffetent tails such that $\sum_{(v_i,v_j)\in T} 2(1 - w^*_{ij}) < 2$ and $\sum_{(v_i,v_j)\in T} d_{ij} \leq d_0$

$S = \emptyset$
for all $v_j \in V\backslash(V(T) \cup \{v_1\})$ **do**
  $l_1(v_j \mid S) := 2(y^*_{jj} + 1)$
  $nearest(v_j \mid S) := \arg\min\{d_{jk} : v_k \notin S \cup \{v_j\}\}$
  $l_2(v_j \mid S) := d_{j,nearest(v_j|S)}$
  $score(v_j \mid S) := l_1(v_j \mid S)/l_2(v_j \mid S)$
end for
while not exit and $V\backslash(V(T) \cup S \cup \{v_1\})$ not empty **do**
  sort vertices in $V\backslash(V(T) \cup S \cup \{v_1\})$ by incresing value of score
  $S := S \cup \{$first vertex in the sorted list$\}$
  if $x^*(\delta(S)) + \sum_{(v_i,v_j)\in T} 2(1 - w^*_{ij}) < 2$ and $\sum_{v_i\in S} \min_{v_j\notin S} d_{ij} + \sum_{(v_i,v_j)\in T} d_{ij} > d_0$ **then**
    S and T define a violated cover-connectivity constraint
    exit
  else
    for all $v_j \in V\backslash(V(T) \cup S \cup \{v_1\})$ **do**
      $l_1(v_j \mid S) := 2\left(y^*_{jj} - x^*\left(\delta(j) \cap \delta(S)\right) + 1\right)$
      $l_2(v_j \mid S) := \sum_{v_i\in S\cup\{v_j\}} \min_{v_k\notin S\cup\{v_j\}} d_{ik} - \sum_{v_i\in S} \min_{v_k\notin S} d_{ik}$
      $score(v_j \mid S) := l_1(v_j \mid S)/l_2(v_j \mid S)$
    end for
  end if
end while

Table 5.1 summarizes the information about the facet results and the separation problems concerning the cuts that appear in our Branch-and-Cut scheme.

Name	Form	Facet	Separation
GSEC	$x(\delta(S)) \geq 2 \sum_{v_j \in S} y_{ij}$	yes Prop. (21)	Exact, polynomial time
GSEC with $\|S\| = 2$	$x_{ij} + y_{ij} \leq y_{jj}$	yes Prop. (21)	Exact, polynomial time
2-matching	$x(E(H)) + x(T) \leq \sum_{v_i \in H} y_{ii} + \frac{\|T\|-1}{2}$	yes Prop. (23)	Heuristic, polynomial time
Cover-connectivity	$x(\delta(S)) \geq 2\Big(\sum_{(v_i,v_j)\in T} y_{ij} - (\|T\| - 1)\Big)$	? Prop. (14)	Heuristic, $\mathcal{NP}$-hard

Table 5.1: Summary of cuts

5.2 Branch-and-Cut implementation

We now outline the main ingredients of our Branch-and-Cut algorithm for both MCP1 and MCP2, pointing out the difference between both problems only when it is necessary.

5.2.1 Preprocessing

Before starting with the optimization we try to fix the value of some variables using logical implications. We have used a very easy preprocessing, which consists of setting to 1 all y_{ii} variables corresponding to vertices v_i such that $\min\{d_{ij} : v_j \in V \backslash \{v_i\}\} \geq d_0 + 1$. This preprocessing procedure applies only to MCP2.

5.2.2 Initialization

To start the optimization we have to define the constraints of the first linear program (LP).

MCP1

For MCP1 our choice was to take the degree constraints, the assignment constraints, the constraints that force v_1 to be in the solution and the logical constraints involving v_1. This gives the following formulation, involving $4|V| - 2$ constraints:

$$\text{minimize} \quad \sum_{[v_i,v_j]\in E} c_{ij}x_{ij} + \sum_{(v_i,v_j)\in A} d_{ij}y_{ij}$$

subject to:

$$\begin{aligned}
x(\delta(i)) &= 2y_{ii} && \text{for all } v_i \in V \\
\sum_{v_j\in V} y_{ij} &= 1 && \text{for all } v_i \in V \setminus \{v_1\} \\
y_{11} &= 1 && \\
y_{1j} &= 0 && \text{for all } v_j \in V \setminus \{v_1\} \\
x_{1j} &\leq y_{jj} && \text{for all } v_j \in V\setminus\{v_1\}
\end{aligned}$$

MCP2

The initial LP for MCP2 differs from that of MCP1 in the objective function and in the knapsack constraint. It is defined as:

$$\text{minimize} \quad \sum_{[v_i,v_j]\in E} c_{ij}x_{ij}$$

subject to:

$$\begin{aligned}
x(\delta(i)) &= 2y_{ii} && \text{for all } v_i \in V \\
\sum_{v_j\in V} y_{ij} &= 1 && \text{for all } v_i \in V \setminus \{v_1\} \\
y_{11} &= 1 && \\
y_{1j} &= 0 && \text{for all } v_j \in V \setminus \{v_1\} \\
x_{1j} &\leq y_{jj} && \text{for all } v_j \in V\setminus\{v_1\} \\
\sum_{\substack{(v_i,v_j)\in A \\ v_i\neq v_j}} d_{ij}y_{ij} &\leq d_0 &&
\end{aligned}$$

Moreover, it is possible to determine a lower bound n_0 on the number of vertices in the solution as follows. Sort the vertices in increasing order of the value $l(i) := \min\{d_{ij} : v_j \in V \backslash \{v_i\}\}$. Let k be the index in the sorted list such that $\sum_{i=0}^{k-1} l(i) \leq d_0$ and $\sum_{i=0}^{k} l(i) > d_0$. Then $n_0 := \max\{3, n-k+1\}$. This results in the constraint

$$\sum_{v_i \in V} y_{ii} \geq n_0, \tag{5.1}$$

which is also added to the initial linear program, making up a total of $4|V|$ constraints.

5.2.3 Feasibility check

A solution of the LP-relaxation of the MCP is feasible if all variables have values zero or one, there are not subtours and each time a vertex v_i is assigned to a vertex v_j, v_j is in the tour. (Note that for MCP2, the knapsack constraint is always in the LP an so it is satisfied by all LP-solutions.)

We start by checking that all y^*_{ij} variables, $i \neq j$, have 0-1 values, and that $y^*_{ij} \leq y^*_{jj}$.

To check if the graph induced by the edge variables with value one is connected we use a disjoint set structure. Initially each vertex is a set by itself. Then we scan all edges. If there is an edge with value one, we check if its two end vertices are contained in disjoint sets. If this is the case, we merge the two sets and continue. Otherwise, the LP-solution contains a subtour, except if the edge between the two vertices is the last edge closing the tour.

5.2.4 Cutting plane phase

Given a fractional solution, we apply the separation procedures described in Section 5.1 in the following order, fixed after performing several computational experiments:

MCP1

(i) two-matching inequality separation (Section 5.1.2),

(ii) separation of generalized subtour elimination constraints with $|S| > 2$ (Section 5.1.1),

(iii) complete enumeration of generalized subtour elimination constraints with $|S| = 2$ (3.15).

MCP2

(i) heuristic 1 with $S = \emptyset$ for extended cover inequalities (3.26),

(ii) two-matching inequality separation (Section 5.1.2),

(iii) heuristic 2 with $T = \emptyset$ for inequalities (3.23),

(iv) heuristic 2 for cover-connectivity inequalities (Section 5.1.3),

(v) heuristic 1 for cover-connectivity inequalities (Section 5.1.3),

(vi) separation of generalized subtour elimination constraints with $|S| > 2$ (Section 5.1.1),

(vii) complete enumeration of generalized subtour elimination constraints with $|S| = 2$ (3.15).

For both MCP1 and MCP2, all inequalities are global (i.e., valid in all the Branch-and-Cut tree). At each iteration we allow a maximum of 300 violated cuts to be generated; if this number is reached we stop the separation phase, introduce those inequalities in the LP-relaxation and reoptimize it. Each five iterations we call the LP-based heuristic with the current (fractional) solution. When the current solution is fractional but no separation procedure finds a violated cut then branching is applied.

Pool separation is discarded since, from our computational experience, we deduce that, for this particular problem, direct separation is more efficient than scanning the pool data structure.

5.2.5 Branching strategy

Our first branching strategy consists of selecting a y_{ii} variable with fractional value. To this end, we applied the "strong branching" rule (see Jüenger and Thienel [39] for details) within the five y_{ii} variables with fractional value closest to 0.5. If all y_{ii} variables have integer values, we select a x_{ij} variable using the same criterion. The last choice would be to branch on a y_{ij} variable with $v_i \neq v_j$.

5.2.6 Heuristic procedures

We have implemented two heuristics. The first one (initial heuristic) gives a feasible solution, and is executed at the beginning of the overall Branch-and-Cut algorithm. The second one (LP-based heuristic), is based on the current fractional solutions, and it is executed each five cutting-plane iterations.

Initial heuristic

This heuristic is a constructive method that generates a feasible solution to be used as first global upper bound in the Branch-and-Cut scheme. We start with the depot and, in each iteration a new vertex is added until feasibility is reached, that is, until $\sum_{v_k \notin V(C)} \min\{d_{kj} : v_j \in V(C)\} \leq d_0$ (being $V(C)$ the sets of vertices in the solution C).

When a new vertex is added to the cycle, two things happen: the length of the cycle increases and the total assignment cost decreases. In order to take into account these two facts, our addition criterium will be the minimization of the convex combination of the increment in the length and (minus) the decrement in the total assignment cost. That is, each time we chose to add to the cycle the vertex $v_i \notin V(C)$ that minimizes

$$f(i, \lambda) := \lambda \, \Delta Len_i + (1 - \lambda)(-\Delta Cos_i),$$

being ΔLen_i and ΔCos_i the increment in the length and the decrement in the total assignment cost, respectively, produced when vertex $v_i \notin V(C)$ is added to the cycle C, and being $0 \leq \lambda \leq 1$. The value of λ represents the relative importance given to the increment of length and the decrement of

assignment cost. If $\lambda = 1$, the priority is that the addition of the next vertex produces the smallest possible increment in the length of the cycle. If $\lambda = 0$, the criterium is to choose the vertex that gives the biggest decrement in the total assignment cost when added to the cycle. Algorithm 5 gives a detailed description of this procedure.

Algorithm 5 Constructive procedure for MCP2

Data: A graph $G = (V, E)$, and the values of d_0 and λ.

$V(C) := \{v_1\}$
for all $v_i \neq v_1$ **do**
  $f(i, \lambda) := \lambda\, c_{1i} + (1 - \lambda)(-\sum\{d_{j1} - d_{ji} : v_j \neq v_1 \text{ and } d_{j1} > d_{ji}\})$
end for
add to the cycle the vertex v_i that minimices $L(i, \lambda)$
repeat
  for all $v_i \notin V(C)$ **do**
    $\Delta Len_i := \min\{c_{si} + c_{ti} - c_{st} : [v_s, v_t] \in C\}$
    $s_i := s$
    $t_i := t$
    $\Delta Cos_i := \sum\{\min_{v_k \in V(C)} d_{jk} - d_{ji} : v_j \notin V(C) \text{ and } \min_{v_k \in V(C)} d_{jk} > d_{ji}\}$
    $f(i, \lambda) := \lambda \Delta Len_i + (1 - \lambda)(-\Delta Cos_i)$
  end for
  select the vertex the vertex v_i that minimices $f(i, \lambda)$
  add v_i to the cycle in the best position, that is, between v_{s_i} and v_{t_i}
until $\sum_{v_k \notin V(C)} \min\{d_{kj} : v_j \in V(C)\} \leq d_0$

If $d_0 = 0$, we only test with $\lambda = 0.1$, otherwise we test with $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$ and take the best solution identified. Once the vertices of the cycle have been selected in this fashion, a better Hamiltonian cycle is sought by means of a 2-opt procedure.

The method we have just described finds a feasible solution for MCP2. To adapt it for MCP1 the only thing we need to change is the stopping rule. Before adding the selected vertex we evaluate the cost of the solution (cycle length plus total assignment cost). The addition is performed only if the new solution is better than the former one; otherwise, the process stops. See algorithm 6 for a detailed description.

Algorithm 6 Constructive procedure for MCP1

Data: A graph $G = (V, E)$, and the value of d_0.

$V(C) := \{v_1\}$
for all $v_i \neq v_1$ **do**
  $f(i, \lambda) := \lambda\, c_{1i} + (1 - \lambda)(-\sum\{d_{j1} - d_{ji} : v_j \neq v_1 \text{ and } d_{j1} > d_{ji}\})$
end for
add to the cycle the vertex v_i that minimices $f(i, \lambda)$
$length := 2c_{1i}$
$assign := \sum_{j \notin \{1,i\}} \min\{d_{j1}, d_{ji}\}$
$newcost := length + assign$
repeat
  $cost := newcost$
  for all $v_i \notin V(C)$ **do**
    $\Delta Len_i := \min\{c_{si} + c_{ti} - c_{st} : [v_s, v_t] \in C\}$
    $s_i := s$
    $t_i := t$
    $\Delta Cos_i := \sum\{\min_{v_k \in V(C)} d_{jk} - d_{ji} : v_j \notin V(C) \text{ and } \min_{v_k \in V(C)} d_{jk} > d_{ji}\}$
    $f(i, \lambda) := \lambda \Delta Len_i + (1 - \lambda)(-\Delta Cos_i)$
  end for
  select the vertex the vertex v_i that minimices $f(i, \lambda)$
  $length := length + \Delta Len_i$
  $assign := assign - \Delta Cos_i$
  $newcost := length + assign$
  if $newcost < cost$ **then**
    add v_i to the cycle in the best position, that is, between v_{s_i} and v_{t_i}
  end if
until $newcost \geq cost$

LP-heuristic

The aim of this heuristic is to improve the upper bound by considering the information given by the current fractional solution (x^*, y^*). To this purpose, we consider value x^*_e to be, in some sense, the probability of edge e to be in the solution.

MCP1 The method we propose works as follows. Given a fractional solution (x^*, y^*), the edges are sorted by decreasing value of x^*_e. They are taken in that order until the resulting subgraph contains a cycle C. The subgraph is then pruned to reduce it only to C. If v_1 is not visited by C, we introduce it in the way that gives the smallest possible increment in the cost of the cycle.

MCP2 The first part of the heuristic coincides with the one for MCP1. If the cycle C obtained that way does not satisfy the knapsack constraint (3.20), it is enlarged by adding a vertex that produces the smallest increment in the cost of the cycle when added to it. This enlargement step is repeated as many times as necessary to get a feasible solution.

Chapter 6

Variable Neighbourhood Tabu Search heuristic

In this chapter we describe a new metaheuristic called VNTS, and show how it can be applied to solve MCP.

6.1 The VNTS Heuristic

Local Search methods for combinatorial and global optimization proceed by performing a sequence of local changes in an initial solution which improve each time the value of the objective function until a local optimum is found. That is, at each iteration an improved solution x' in the neighbourhood $\mathcal{N}(x)$ of the current solution x is obtained, until no further improvement is found. In recent years, several metaheuristics have been proposed which extend in various ways this scheme and avoid being trapped in a local optimum. The most famous of these metaheuristics are Simulated Annealing (Kirkpatrick et al. [44]), Genetic Search (Holland [35]), and Tabu Search (Glover [27] and [28]).

To avoid termination at a local minimum, the metaheuristics allow non-improving moves. However, this presents the risk of cycling. In Tabu Search the use of a short term memory helps to forbid moves that might lead to recently visited solutions.

Variable Neighbourhood Search (Mladenovic [53], Mladenovic and Hansen [54]) is a relatively new technique whose originality consists of trying to escape from local optima by changing the neighbourhood structure. In its basic form, VNS explores a set of neighbourhoods of the current solution, makes a local search from a neighbour solution chosen at random to a local optimum, and moves to it only if there has been an improvement. Variable Neighbourhood Descent (VND) is a variant of VNS in which the change of neighbourhoods is performed during the local search phase. Let us denote a finite set of pre-selected neighbourhood structures with $\mathcal{N}_k$, $k = 1, ..., k_{\max}$, and with $\mathcal{N}_k(x)$ the set of solutions in the k-th neighbourhood of x. The scheme of VND is the following.

VND algorithm

1. Initialization:
   (a) Select a set of neighbourhoods $\mathcal{N}_k$, $k = 1, ..., k_{\max}$, that will be used in the descent.
   (b) Find an initial solution x.
2. Repeat the following until no improvement is obtained:
   (a) Set $k := 1$.
   (b) Repeat the following steps until $k = k_{\max}$:
      i. Exploration of the neighbourhood: find the best neighbour x' of x ($x' \in \mathcal{N}_k(x)$).
      ii. Move or not: if x' is better than x, set $x := x'$. Otherwise, set $k := k + 1$.

end

The metaheuristic method we propose, the Variable Neighbourhood Tabu Search (VNTS), was originally inspired by VND. However, our construction of the neighbourhoods is somehow different, since in VNTS the different neighbourhoods used are nested and are defined from an original one $\mathcal{N}$ in the following way:

$$\mathcal{N}_1(x) := \mathcal{N}(x),\ \mathcal{N}_2(x) := \bigcup_{y \in N_1(x)} \mathcal{N}(y), ...,\ \mathcal{N}_k(x) := \bigcup_{y \in N_{k-1}(x)} \mathcal{N}(y).$$

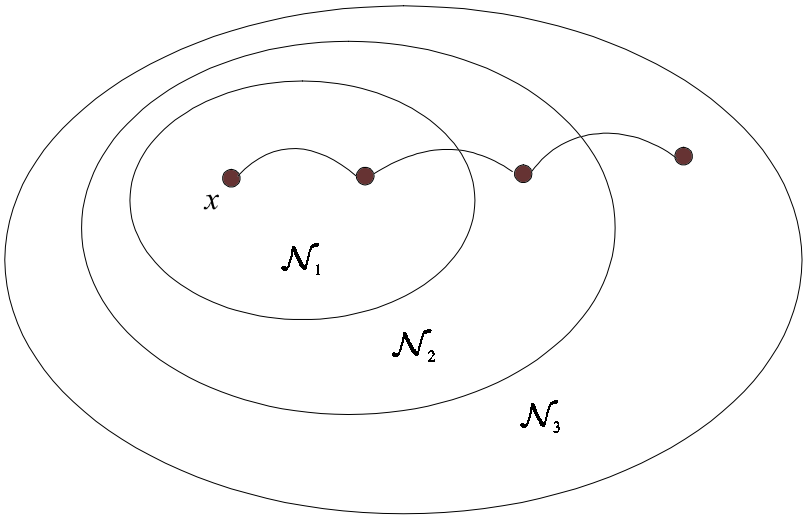

Figure 6.1: Neighbourhood structures in VNTS

That is, $\mathcal{N}_i(x)$ contains the neighbours of x at distance 1 to i (see Figure 6.1).

The VNTS comprises the following steps:

VNTS algorithm

1. Initialization:
   (a) Select an initial neighbourhood $\mathcal{N}$, from which $\mathcal{N}_k$, $k = 1, ..., k_{\max}$ are defined as described before.
   (b) Find an initial solution x.
   (c) Set $best\_x := x$.
   (d) Choose a stopping condition.
2. Repeat the following until the stopping condition is met:
   (a) Local Search: apply a tabu local search in $\mathcal{N}(x)$ until a local minimum x' is found.
   (b) If x' is better than $best\_x$, do $best\_x := x'$.

(c) Shaking: generate at random a point x'' in $\mathcal{N}_s(x')$.

(d) Set $x := x''$.

end

Tabu search tools are incorporated to both local search and shaking procedures to modulate the intensification and diversification of the search. The local search tests all the nontabu moves, and that providing the best feasible solution is made. The shake consists of applying a number of random moves (feasible or not). This number of random moves is the size of the shake. The shake could provide an infeasible solution without feasible neighbour. In those cases the local search is allowed to make the best nontabu move among those that reduce the infeasibility. Therefore, the local search consists of choosing the best feasible nontabu move, if there is one, or the best nontabu move that reduces the infeasibility. As stopping criteria usual conditions can be applied: the total CPU time, the number of local searches, the number of moves, the relative number of moves without improving the best solution found.

The tabu tool proposed is a list with the last elements that have been dropped from the solution. The elements that are in the tabu list are not allowed to go immediately inside the solution again. This tabu tool is more powerful than the usual list of tabu moves, since with a short list of tabu elements many moves are forbidden.

To show the wide applicability of the VNTS metaheuristic consider the problems where each solution is given by a selection of a set of elements from an universe. Many of relevant combinatorial optimization problems can be formulated in this way. The standard moves for those problems are the add, drop and the interchange moves. These moves consist of adding a new member to the solution, dropping a member from the solution, or interchanging a member of the solution for another one out of the solution. The selection can be subject to some constraints. In those cases, the moves that provide a feasible solution are called feasible moves. In next section we mention some aspects to be taken into account when applying VNTS to a general Location-Allocation problem.

6.2 General aspects of the application of VNTS to Location-Allocation problems

The Location-Allocation problems are those consisting of selecting a set of points S to hold the facilities, and allocating the users to them. A solution S is usually evaluated by two functions that we will call *length*, $Len(S)$, and *cost*, $Cos(S)$, of the solution. The length of the solution represents the location cost, that can be defined as the number of points selected, the total length of a cycle, of a tree, or any other measure of the structure connecting the points in the solution, etc. The cost of the solution S is a measure of the structure connecting the users to the set of selected points. Usual cost functions are the sum or the maximum of the distances from the users to the points in the solution.

The most common approaches used to solve this kind of bicriteria problems are to fix an upper bound on one of the two functions and to minimize the other one, or to minimize a convex combination of them.

6.2.1 Coding of the solutions

The selection of a solution coding that provides an efficient way of implementing the moves and evaluating the solutions is essential for the success of any search method.

We propose the following coding of the solutions. Let n be the size of the universe. A solution S is represented by an array $[s_i : i = 1, ..., n]$ where s_i is the i-th element of the solution, for $i = 1, 2, ..., m$, and the $(i - m)$-th element outside the solution, for $i = m + 1, ..., n$.

6.2.2 Evaluation of the solutions

For some problems, the evaluation of a solution could mean to solve another optimization problem. For instance, when looking for routes the solutions are ordered sets of visited points. Therefore, the evaluation of the length of the shortest route requires to solve a TSP. The efficiency when evaluating the modification produced in the objective function by a move is very important, because a great deal of moves are tested. Therefore, having an efficient

procedure to (may be heuristically) evaluate the modifications of the solution is very useful. The information given by the actual solution can be used to design an efficient way of evaluating a move, by considering only the local modifications that it would produce. So, when a solution is modified by adding and/or dropping a point, the length of the new solution is obtained by extracting the leaving point and/or looking for the best position to insert the new one. In addition, to calculate the cost of the new solution we need to know the new optimal allocation of the users. Usually we only need to know if the entering point can improve the allocation of the users and if the users allocated to the leaving point need to be reallocated.

6.2.3 The moves

The usual moves considered in this kind of problems are the add, drop and interchange moves. Given a solution S and an element s_j not in the solution, an add move consists of adding s_j to S. A drop move consists of eliminating an element s_i from S. Given a solution S, an element s_i in the solution and an element s_j not in the solution, an interchange or add/drop move consists of dropping s_i from S and adding s_j to S.

If some constraint is imposed, not every move is feasible. The moves that provide a feasible solution are called feasible moves. If the constraint consists in an upper bound on a function of the solution then the moves that reduce the infeasibility can be used to get feasible solutions.

6.2.4 Initial solution

Usually a constructive heuristic is used to generate an initial solution. Standard constructive heuristics consists of a series of add moves starting from an empty solution. During the building process the length of the candidate solutions increases while their allocation cost decreases. The procedure stops when the solution obtained is feasible. The more common strategies for adding elements to the solutions are to use simple, random, and good add moves. The good properties that a constructive method should have in order to be used in a metaheuristic are:

Quality: The degree of optimality of the solution proposed must be high.

Efficiency: The amount of resources (time) used must be small.

Randomness: The ability to cover the whole search space.

The following procedures are standard constructive methods:

1. **Random Insert method**. Select a new member at random.
2. **Best Insert method**. Select a new member in order to minimize the increment in the length of the solution.
3. **Cheapest Insert method**. Select a new member in order to maximize the decrement in the cost of the new solution.

6.3 The VNTS for MCP

In this section we describe with more details the implementation of a VNTS heuristic for MCP.

6.3.1 Coding of the routes

The solution given by a cycle C visiting m vertices is represented by an array $S = [v_i : i = 1, ..., n]$ where v_1 is the depot, and v_i is:

- the i-th vertex of the cycle, for $i = 2, ..., m$, and
- the $(i - m)$-th vertex not belonging to the cycle, for $i = m + 1, ..., n$.

Note that the solution gives not only the points visited but also their order in the cycle, starting from the depot.

The assignment or allocation costs d_{ij} are stored in a matrix D, and the lengths or routing costs c_{ij} are stored in a matrix L. The length of the solution cycle represented by S is given by:

$$Len(S) := \sum_{i=2}^{m} L[v_{i-1}, v_i] + L[v_1, v_m].$$

And its allocation cost is given by:

$$Cos(S) := \sum_{i=m+1}^{n} \min_{j=1..m} D[s_i, s_j].$$

An optimal solution S^* of MCP1 is a solution that minimizes $Len(S) + Cos(S)$. The solution S is feasible for MCP2 if $Cos(S) \leq d_0$. An optimal solution S^* for MCP2 is a feasible solution that minimizes $Len(S)$, that is, the solution S^* such that $Cos(S^*) \leq d_0$ and $Len(S^*) := \min\{Len(S) : Cos(S) \leq d_0\}$.

6.3.2 The moves

In this section we show how to recompute the length and cost of a solution after a move has been done.

Inserting a vertex in the cycle

The insertion of a new vertex v_j in the cycle implies to know in which position it is to be inserted. The assignment cost of the new solution does not depend on the insertion position. So we will always insert the new vertices in the best possible position, i.e., the position that provides a shortest cycle.

The length of the new solution S_{kj}, obtained by the insertion of a new vertex v_j in the cycle just after the vertex v_k, is given by:

- for $k = 1, ..., m-1 : Len(S_{kj}) := Len(S) + L[v_k, v_j] + L[v_j, v_{k+1}] - L[v_k, v_{k+1}]$,
- for $k = m : Len(S_{kj}) := Len(S) + L[v_m, v_j] + L[v_j, v_1] - L[v_m, v_1]$,

This can be rewritten as:

$$Len(S_{kj}) := Len(S) + L[v_k, v_j] + L[v_j, v_{k \bmod m+1}] - L[v_k, v_{k \bmod m+1}].$$

Every vertex is inserted at the position k where the minimum

$$\min_{k=1..m} Len(S_{kj})$$

is reached. Therefore, to evaluate the length of the new cycle takes $O(m)$ time.

After the insertion, the new total cost is given by:

$$Cos(S_{kj}) := Cos(S) - \sum_{i=m+1}^{n} \max\left\{0, \min_{h=1..m} D[v_i, v_h] - D[v_i, v_j]\right\}.$$

Note that the decrement in the cost depends only on the vertex inserted, and not on the position k.

Improving the updating of the costs The computation of the cost of the new solution after each adding move can be simplified by storing the individual allocation costs in a vector named $Cos1[.]$, defined as:

$$Cos1[i] := \min_{j=1..m} D[v_i, v_j],\ i = 1, ..., n.$$

Therefore, the total cost is:

$$Cos(S) := \sum_{i=m+1}^{n} Cos1[i].$$

The new individual allocation costs obtained when the vertex v_j is inserted can be calculated in the following way:

$$Cos1[i] := \min\{Cos1[i], D[v_i, v_j]\}.$$

Then, to calculate the new allocation cost takes $O(n)$ time.

Deleting a vertex from the cycle

Let S_k denote the solution obtained if the k-th vertex of the cycle, v_k, is removed from S, for $k = 1, ..., m$. The new length, depending on k, is given by:

$$Len(S_k) := Len(S) + L[v_{k-1}, v_{k \bmod m+1}] - L[v_{k-1}, v_k] - L[v_k, v_{k \bmod m+1}]$$

The individual allocation costs have to be recomputed only for the vertices v_i such that $Cos1[i] = D[v_i, v_k]$. For those vertices,

$$Cos1[i] := \min_{\substack{j=1,...,m \\ j \neq k}} D[v_i, v_j].$$

Second Improvement in the updating of the costs The computation of the cost of the new solution obtained when a vertex is removed from the cycle can be improved by using the second best allocation. Consider a solution S in which the individual allocation costs for every vertex v_i, $i = 1, ..., n$, is

$$Cos1[i] := \min_{j=1,...,m} D[v_i, v_j] = D[v_i, v_{r(i)}].$$

Then let the second best allocation cost of v_i, $i = 1, ..., n$, be

$$Cos2[i] := \min_{\substack{j=1,...,m \\ j \neq r(i)}} D[v_i, v_j].$$

Let V_k be the set of vertices allocated to vertex v_k, that is, $V_k := \{v_i \in V : Cos1[i] = D[v_i, v_k]\}$. Then, if vertex v_k is removed from the cycle, the new total allocation cost is:

$$Cos(S_k) := \sum_{v_i \in V_k} Cos2[i] + \sum_{v_i \in V \setminus V_k} Cos1[i].$$

So, the new allocation cost can be computed in $O(n)$ time.

When removing v_k, the first and second individual allocation costs have to be recomputed only for the vertices v_i such that $D[v_i, v_k] \leq Cos2[i]$. For those vertices v_i, the first allocation cost are updated using the following rule:

$$\text{if } Cos1[i] = D[v_i, v_k], \text{ then } Cos1[i] := Cos2[i].$$

And the second allocation costs are updated by

$$Cos2[i] := \min_{\substack{j=1,...,m \\ j \neq r(i)}} D[v_i, v_j],$$

where $r(i)$ is such that $Cos1[i] = D[v_i, v_{r(i)}]$.

The second best allocation cost is a tool similar to the one used in the VND by Hansen, Mladenovic, and Pérez-Brito [34].

Interchanging two vertices

The interchange of two vertices consists of a combination of an add and a drop move. A vertex is removed from the cycle, and a vertex is inserted

in the best possible position in the cycle. The length and cost of the new solution are computed as follows. Let S_{ij} be the new solution consisting in interchanging v_i and v_j, by dropping v_i and adding v_j in the best possible position, say after k. Then the new length is:

$$\begin{aligned} Len(S_{ij}) := \quad & Len(S)+ \\ & L[v_{i-1}, v_{(i+1) \bmod m+1}] - L[v_{i-1}, v_i] - L[v_i, v_{(i+1) \bmod m+1}]+ \\ & \min_{k=1,\ldots,m} \left\{ L[v_k, v_j] + L[v_j, v_{(k+1) \bmod m+1}] - L[v_k, v_{(k+1) \bmod m+1}] \right\}. \end{aligned}$$

So it takes $O(m)$ time to evaluate the new cycle length.

The new allocation cost is given by:

$$Cos(S_{ij}) := \sum_{k=m+1}^{n} \min\{D[v_k, v_j], \min_{\substack{l=1,\ldots,m \\ l \neq i}} D[v_k, v_l]\},$$

which implies $O(m \cdot n)$ operations. However, using the values in $Cos1$ and $Cos2$, an improved procedure can be applied. We first test if the going in vertex is the best or the second best allocation for the users. This is done for every $k > m$, in the following way: if $D[v_k, v_j] \leq Cos1[k]$, then $Cos2[k] := Cos1[k]$ and $Cost1[k] := D[v_k, v_j]$. Otherwise, if $D[v_k, v_j] \leq Cost2[k]$ then $Cos2[k] := D[v_k, v_j]$.

Then we need to update the individual costs of the users assigned to the dropped vertex, i.e., for the users v_k, $k > m$, such that $Cos1[k] = D[v_k, v_i]$. For those v_k we do $Cos1[k] := Cos2[k]$ and

$$Cos2[k] := \min_{\substack{l=1,\ldots,m \\ l \neq r(k)}} D[v_k, v_l],$$

where $r(k)$ is such that $Cos1[k] = D[v_k, v_{r(k)}]$.

The updating of the individual allocation costs, and therefore, of the total allocation cost, takes $O(mn_i + n)$ time, n_i being the number of vertices assigned to vertex v_i.

6.3.3 The initial solution

To obtain an initial cycle we use a constructive method. We start only with the depot. At each iteration a new vertex is chosen at random at it is inserted

in the best position, i.e., the position that gives a smallest increment in the value of the objective function. At each iteration the computation of the length takes $O(m)$ time and the computation of the cost takes $O(mn)$. The procedure stops when there are at least three vertices in the cycle and:

- For MCP1: the value of the objective function cannot be decreased anymore.

- For MCP2: the cycle becomes feasible.

6.3.4 Tabu list

We use a tabu list to improve the efficiency of the search by avoiding moves leading to recently visited solutions. In order to do so we put in a FIFO tabu list the vertices dropped from the cycle. The tabu list is kept at the end of the array S that codes the solution. In fact, if the size of the tabu list is t, a solution consisting of a cycle visiting m vertices is represented by an array $S = [v_i : i = 1, ..., n]$ where v_1 is the depot, and v_i is:

- the i-th vertex of the cycle, for $i = 2, ..., m$,
- a vertex outside the cycle, for $i = m + 1, ..., n$, and
- a vertex in the tabu list, for $i = n - t + 1, ..., n$.

When a vertex is removed from the cycle it is first placed in a position k, being k the position $m + 1$ if the performed move was a drop move, and the position previously occupied by the added vertex if an add-drop was performed. In a second step, the vertex in position k and the first vertex in the tabu list are interchanged .

Only vertices v_j, with $j = m + t + 1$ to n, are tested for being included in the cycle. So, a leaving vertex will be out of the cycle in at least t iterations (the size of the tabu list). Note that this tabu tool is more powerful than the usual list of tabu moves, since with a short list of tabu elements many moves are forbidden.

6.3.5 Shake procedure

The shake procedure (used by Mladenovic and Hansen [54] in their VNS heuristic) allows to escape from local minimum without destroying completely the good properties of the current solution. Given a size s for the shake procedure, we choose s times two vertices at random, v_i and v_j. If the vertex v_i is in the cycle and v_j is outside the cycle we do the corresponding add/drop move, if both vertices are in the cycle we drop v_i, and if both vertices are outside the cycle we add v_j. The going-in vertex v_j is always chosen taking into account the tabu list, and it is inserted in the best possible position. The going-out vertex v_i, if dropped, goes to the tabu list, that is updated.

6.3.6 Local search

Given a solution, the greedy local search is implemented by choosing each time the best possible move among all the add, drop or add/drop moves. For MCP1 the best move is that minimizing the sum of length and allocation costs. For MCP2, the best possible move is that with minimum length among those that provide a feasible cost or a less infeasible cost (the latest only if there are not feasible moves). This is done to allow the process to move through feasibility if this property has been lost when making the shake.

Let S_{ij} denote the solution obtained from S by interchanging v_i and v_j, for $i = 1, ..., m$ and $j = m+1, ..., n$. Let S_k denote the solution obtained from S by dropping v_k from the cycle, if $k = 1, ..., m$, or adding v_k to it, if $k = m+1, ..., n$. The pseudocode of a local searches is:

Local Search for MCP1

Start with S^n, $n = 0$.

Repeat

$$S^{n+1} \leftarrow \arg\min \begin{cases} \arg \min\limits_{k=1,...,n} \{Len(S_k^n) + Cos(S_k^n)\} \\ \arg \min\limits_{\substack{i=1,..,m \\ j=m+1,...,n}} \{Len(S_{ij}^n) + Cos(S_{ij}^n)\} \end{cases}$$

$n \leftarrow n + 1$

Until $Len(S^{n-1}) + Cos(S^{n-1}) = Len(S^n) + Cos(S^n)$.

Local Search for MCP2

Start with S^n, $n = 0$.

Repeat

$$S^{n+1} \leftarrow \arg\min \begin{cases} \arg \min\limits_{k=1,\ldots,n} \{Len(S_k^n) : Cos(S_k^n) \leq \max\{d_0, Cos(S^n)\} \\ \arg \min\limits_{\substack{i=1,..,m \\ j=m+1,\ldots,n}} \{Len(S_{ij}^n) : Cos(S_{ij}^n) \leq \max\{d_0, Cos(S^n)\} \end{cases}$$

$n \leftarrow n + 1$

Until $Len(S^{n-1}) = Len(S^n)$.

After the set of vertices in the solution has been determined this way, the length of the cycle is improved by performing classical 2-opt interchanges of edges (see Lin and Kernighan [48]).

6.3.7 The strategy for the sizes

The size of t of the tabu list is initially 0. Each time a vertex is removed from the cycle it goes to the tabu list and t augment one unit. When t reaches a given bound (ten per cent of n in our case) it is fixed, and the tabu list starts to work as a FIFO list.

The parameter s that control the shake procedure is initialized to 2. Each time a local minimum is reached the shake procedure is applied s times. If the current local minimum is equal to the one found just before, s is augmented by one unit. If the current local minimum is better than the best know solution, s is set again to 2. We do not allow s to become larger than m.

6.3.8 Stopping criterion

As stopping criterion we use the number of iterations without improving the best solution found. Let it be the current iteration and it^* be the iteration where the best solution was found, then we stop if $it - it^* > 25$.

Chapter 7

Computational results

In this chapter we present the computational results obtained when solving MCP by the Branch-and-Cut and the VNTS algorithms described in the former two chapters. We also compare the VNTS results with those given by other heuristic methods.

All the computational experiments reported were performed on a SUN Ultra-60 computer running at 300 MHz, with the following specifications: 95 SPECs, 13 MIPs, and 23.5 MFlops. Regarding the test instances, the depot was always chosen as the first vertex. For MCP1, the assignment and routing costs were symmetric and proportional to the distance between the vertices considered. Let us denote by l_{ij} the distance between vertices v_i and v_j. To obtain optimal solutions visiting approximately 100%, 75%, 50% and 25% of the total number of vertices in the instances, we set $d_{ij} = \lceil \alpha \ l_{ij} \rceil$ and $c_{ij} = \lceil (10 - \alpha) l_{ij} \rceil$, for $\alpha \in \{3, 5, 7, 9\}$. For MCP2, the assignment and routing costs were both identical to the distance, and the threshold d_0 was computed as a proportion of the assignment cost c_0 of a shortest 3-edge cycle including the depot. This was done to yield optimal routes visiting approximately 100%, 75%, 50% and 25% of the total number of vertices in the instances. We then set $d_0 = \lceil \beta c_0 \rceil$ for $\beta \in \{0.00, 0.08, 0.22, 0.42\}$.

7.1 Branch-and-Cut results

The Branch-and-Cut algorithm described in Chapter 5 was implemented in C++ programming language. ABACUS 2.2 linked with CPLEX 6.0 was used as a Branch-and-Cut framework. See Jüenger and Thienel [39] for details on this software. We developed two versions of the algorithm, one for MCP1 and another for MCP2. The performance of each one was tested on two different classes of test instances.

The first instance class includes all TSP instances from TSPLIB 2.1 (Reinelt [61]) involving up to 200 vertices (problems 'burma14' to 'kroB200'). The second instance class was randomly generated. For each value of n in $\{25, 50, 75, 100, 125, 150, 175, 200\}$, we generated n vertices with coordinates in $[0, 1000] \times [0, 1000]$, and computed the routing cost as their Euclidean distance rounded up to the next integer. For each pair (n, α) (or (n, β), for MCP2) we generated ten instances.

Tables 7.1 to 7.8 summarize the computational behavior of our Branch-and-Cut codes on the various classes of instances. The column headings are defined as follows.

Name: problem name (for instances from TSPLIB);

$|V|$**:** number n of vertices (for random instances);

α / β**:** scale factor described above;

succ: number of instances solved to optimallity over 10 trials (for random instances);

p^* **:** percentage of vertices included in the optimal cycle;

opt: optimal objective value (for instances from TSPLIB);

%-LB: percentage ratio LB/optimum, where LB is the value of the best heuristic solution computed at the root node;

%-UB: percentage ratio UB/optimum, where UB is the upper bound computed at the root node;

pair: number of constraints of type (3.15);

gsec: number of constraints of type (3.16) with $|S| > 2$;

2mat: number of constraints of type (3.18);

cover: number of constraints of type (3.21) (only for MCP2);

sec2: number of constraints of type (3.23) (only for MCP2);

cc: number of constraints of type (3.26) with $T \neq \emptyset$ and $S \neq \emptyset$ (only for MCP2);

nodes: total number of nodes generated (1 means that the problem required no branching);

time: total computing time spent by the Branch-and-Cut code.

The computing times reported are expressed in the format hh:mm:ss, and refer to CPU time. We imposed a time limit of two hours for each run. For the instances exceeding the time limit, we report '>2:00:00' in the *time* column, and compute the corresponding results by considering the best available solution to be optimal. Hence, for the time-limit instances the column %-UB gives an upper bound on the percentage approximation error. Tables 7.1 to 7.3 and 7.4 to 7.6 refer to problems MCP1 and MCP2, respectively, on TSPLIB instances. Tables 7.7 and 7.8 refer to the same problems on random instances and contain average results over ten trials.

Results presented in Tables 7.1 to 7.8 indicate that the proposed Branch-and-Cut algorithm can solve instances involving up to approximately 150 vertices within modest computing times. For a given problem size, the most difficult instances tend to be those where relatively few vertices belong to the optimal cycle. This can be explained by the fact that the location component of the problem is then important. At the other extreme, instances in which all vertices belong to the cycle reduce to a TSP which is a relatively easy problem for the values of n considered. The %-LB column indicates that the lower bound developed at the root of the Branch-and-Cut tree is very tight, typically within 0.5% of the optimum. The combination of the primal and LP-based heuristics is also very powerful and usually yields solutions within 5% of the optimum. It seems to perform better when not all vertices belong to the cycle. All valid inequalities developed for the two models are frequently generated within the search tree, the most commonly used being

constraints (3.16), (3.15) and (3.23). The number of nodes in the tree is relatively low, and several instances (in particular the TSPLIB instances for MCP1) are solved without any branching. Finally, MCP1 seems easier to solve than MCP2, as reflected by the larger %-LB values. This translates into more branching and larger computing times when a bound limit on the total assignment cost is considered as a constraint. Overall we were able to solve to optimallity 176 and 160 of the 192 TSPLIB instances for MCP1 and MCP2, respectively. For the random instances, the corresponding figures are 280 and 244 out of 320. Note, that the majority of unsolved cases are instances with $n \geq 150$ for MCP1, and with $n \geq 125$ for MCP2.

name	α	p^*	opt	%-LB	%-UB	pair	gsec	2mat	nodes	time
burma14	3	100,0	9969	100,0	100,0	4	11	0	1	0:00:00
burma14	5	64,3	13870	100,0	100,0	12	36	3	1	0:00:00
burma14	7	42,9	13900	100,0	100,0	39	77	0	1	0:00:00
burma14	8	21,4	12116	100,0	100,0	46	83	0	1	0:00:00
ulysses16	3	100,0	20241	100,0	100,0	4	34	4	1	0:00:00
ulysses16	5	75,0	28890	100,0	100,0	17	84	1	1	0:00:00
ulysses16	7	31,3	23980	100,0	100,0	42	181	0	1	0:00:00
ulysses16	9	18,8	12902	100,0	100,0	38	38	0	1	0:00:00
gr17	3	100,0	6255	100,0	100,0	2	46	2	1	0:00:00
gr17	5	70,6	9155	100,0	100,0	24	53	2	1	0:00:00
gr17	7	47,1	8950	100,0	100,0	44	78	0	1	0:00:00
gr17	9	23,5	4720	100,0	100,0	61	72	0	1	0:00:00
gr21	3	100,0	8121	100,0	100,0	2	18	8	1	0:00:00
gr21	5	85,7	12965	100,0	100,0	28	50	2	1	0:00:00
gr21	7	47,6	14287	100,0	100,0	69	209	0	1	0:00:01
gr21	9	14,3	6783	100,0	100,0	89	89	0	1	0:00:00
ulysses22	3	100,0	20703	100,0	100,0	10	67	8	1	0:00:00
ulysses22	5	77,3	29625	100,0	100,0	31	242	3	1	0:00:01
ulysses22	7	36,4	27328	100,0	100,0	78	353	0	1	0:00:01
ulysses22	9	13,6	15222	100,0	100,0	145	157	0	1	0:00:01
gr24	3	100,0	3816	100,0	100,0	8	12	2	1	0:00:00
gr24	5	79,2	5925	100,0	100,0	28	17	0	1	0:00:00
gr24	7	41,7	6547	100,0	100,0	88	344	0	1	0:00:01
gr24	9	20,8	4177	100,0	100,0	166	220	0	1	0:00:01
fri26	3	100,0	2811	100,0	100,0	2	38	14	1	0:00:00
fri26	5	80,8	4445	100,0	100,0	34	79	2	1	0:00:00
fri26	7	46,2	5114	100,0	100,0	97	288	0	1	0:00:01
fri26	9	11,5	3523	100,0	100,0	189	190	0	1	0:00:01
bayg29	3	100,0	4830	100,0	100,0	4	68	7	1	0:00:01
bayg29	5	69,0	7230	100,0	100,0	34	35	2	1	0:00:00
bayg29	7	37,9	7435	100,0	100,0	112	361	0	1	0:00:01
bayg29	9	10,3	4383	100,0	100,0	209	226	0	1	0:00:01
bays29	3	100,0	6060	100,0	100,0	4	40	28	1	0:00:01
bays29	5	69,0	8615	100,0	100,0	32	130	8	1	0:00:01
bays29	7	48,3	8753	100,0	100,0	95	341	0	1	0:00:01
bays29	9	10,3	5604	100,0	100,0	206	335	0	1	0:00:01
dantzig42	3	100,0	2097	99,9	110,3	20	211	35	5	0:00:03
dantzig42	5	69,0	3315	100,0	100,0	59	1176	51	1	0:00:04
dantzig42	7	45,2	3717	100,0	100,0	168	1125	2	1	0:00:06
dantzig42	9	26,2	2848	100,0	100,0	447	871	0	1	0:00:11
swiss42	3	100,0	3819	100,0	100,0	6	45	3	1	0:00:01
swiss42	5	73,8	6025	100,0	100,0	62	243	28	1	0:00:02
swiss42	7	52,4	6574	100,0	100,0	158	666	0	1	0:00:05
swiss42	9	7,1	3523	100,0	100,0	364	377	0	1	0:00:03
att48	3	100,0	31884	99,8	105,8	10	245	76	5	0:00:05
att48	5	70,8	49700	100,0	100,0	80	923	60	1	0:00:05
att48	7	47,9	54842	100,0	100,0	212	952	0	1	0:00:08
att48	9	12,5	39468	100,0	100,0	552	912	0	1	0:00:18
gr48	3	100,0	15138	99,8	102,8	14	238	66	7	0:00:06
gr48	5	68,8	22810	100,0	100,0	73	1273	15	1	0:00:07
gr48	7	52,1	23025	100,0	100,0	183	581	3	1	0:00:05
gr48	9	18,8	17338	100,0	100,0	490	869	0	1	0:00:17
hk48	3	100,0	34383	100,0	100,0	12	198	3	1	0:00:03
hk48	5	83,3	55275	100,0	100,0	67	962	41	1	0:00:05
hk48	7	41,7	58917	100,0	100,0	199	837	0	1	0:00:07
hk48	9	12,5	41170	100,0	100,0	518	1038	0	1	0:00:22
eil51	3	100,0	1278	100,0	107,5	4	181	79	3	0:00:04
eil51	5	74,5	1995	100,0	100,0	61	42	0	1	0:00:02
eil51	7	33,3	2113	100,0	100,0	230	1073	0	1	0:00:14
eil51	9	9,8	1244	100,0	100,0	507	836	0	1	0:00:11
berlin52	3	100,0	22626	100,0	100,0	12	46	0	1	0:00:02
berlin52	5	78,8	36115	100,0	100,0	86	1081	33	1	0:00:07
berlin52	7	46,2	37376	100,0	100,0	219	708	0	1	0:00:08
berlin52	9	9,6	20361	100,0	100,0	535	833	0	1	0:00:12

Table 7.1: Branch-and-Cut results for MCP1 on TSPLIB instances

name	α	p^*	opt	%-LB	%-UB	pair	gsec	2mat	nodes	time
brazil58	3	100,0	76185	100,0	100,0	18	105	3	1	0:00:04
brazil58	5	69,0	115045	100,0	100,0	103	2096	214	1	0:00:14
brazil58	7	48,3	126807	100,0	100,0	239	1653	1	1	0:00:18
brazil58	9	15,5	83690	100,0	100,0	560	1543	0	1	0:00:38
st70	3	100,0	2025	99,8	103,4	30	495	75	5	0:00:11
st70	5	78,6	3110	99,8	102,4	111	3571	406	5	0:00:46
st70	7	42,9	3402	100,0	100,0	288	2061	0	1	0:00:31
st70	9	24,3	2610	100,0	100,0	799	2033	0	1	0:01:47
eil76	3	100,0	1614	100,0	100,0	6	137	43	1	0:00:08
eil76	5	72,4	2460	99,9	100,6	110	1960	54	5	0:00:32
eil76	7	42,1	2504	100,0	100,0	344	2262	0	1	0:00:58
eil76	9	15,8	1710	100,0	100,0	908	2827	0	1	0:03:46
pr76	3	100,0	324477	98,5	101,6	148	27303	11864	1463	0:43:57
pr76	5	80,3	500395	99,7	106,2	169	5855	1280	101	0:05:39
pr76	7	51,3	555845	99,5	103,9	363	9700	400	41	0:06:23
pr76	9	17,1	424359	100,0	100,0	842	2973	0	1	0:04:46
gr96	3	100,0	163935	99,2	102,7	36	1929	591	57	0:04:43
gr96	5	79,2	252850	100,0	100,0	148	3314	324	1	0:01:41
gr96	7	50,0	273599	100,0	100,0	420	3508	1	1	0:02:20
gr96	9	27,1	232823	100,0	100,0	1247	3902	0	1	0:11:42
rat99	3	100,0	3633	99,9	107,3	10	532	99	5	0:00:41
rat99	5	87,9	5885	100,0	100,0	150	1534	213	1	0:00:45
rat99	7	42,4	6436	100,0	100,0	461	3311	0	1	0:02:23
rat99	9	21,2	5150	100,0	100,0	1280	4325	0	1	0:12:01
kroA100	3	100,0	63846	99,7	100,9	28	822	204	19	0:01:28
kroA100	5	80,0	100785	99,8	100,2	151	4894	263	7	0:02:33
kroA100	7	55,0	115388	100,0	100,0	435	3847	1	1	0:02:27
kroA100	9	21,0	94265	100,0	100,0	1218	3682	0	1	0:12:42
kroB100	3	100,0	66423	99,5	105,4	36	2794	444	43	0:03:30
kroB100	5	77,0	104550	100,0	100,0	145	3154	235	1	0:01:19
kroB100	7	46,0	118111	100,0	100,0	442	3299	1	1	0:02:09
kroB100	9	16,0	93938	100,0	100,0	1304	4210	0	1	0:13:19
kroC100	3	100,0	62247	99,8	102,4	48	1313	267	15	0:01:31
kroC100	5	82,0	99065	100,0	100,0	141	2747	227	1	0:01:04
kroC100	7	50,0	113533	100,0	100,0	442	3853	1	1	0:02:38
kroC100	9	23,0	92894	100,0	100,0	1253	3492	0	1	0:11:40
kroD100	3	100,0	63882	99,8	105,6	40	930	170	9	0:01:15
kroD100	5	82,0	101645	100,0	100,0	145	2413	119	1	0:01:07
kroD100	7	47,0	116849	99,9	101,1	441	4372	9	3	0:03:13
kroD100	9	23,0	92102	100,0	100,0	1269	4541	0	1	0:15:16
kroE100	3	100,0	66204	99,3	104,6	48	2196	1008	49	0:03:56
kroE100	5	77,0	104915	99,9	103,7	165	8112	618	7	0:04:06
kroE100	7	51,0	116471	100,0	100,0	441	2820	7	1	0:01:54
kroE100	9	20,0	96116	100,0	100,0	1242	4882	0	1	0:13:06
rd100	3	100,0	23730	100,0	100,0	26	537	129	1	0:00:30
rd100	5	76,0	37975	99,9	102,3	164	6826	301	9	0:03:07
rd100	7	44,0	40915	100,0	100,0	443	5053	16	1	0:03:17
rd100	9	21,0	31776	100,0	100,0	1269	4032	0	1	0:11:44
eil101	3	100,0	1887	99,8	106,8	18	739	208	19	0:01:16
eil101	5	72,3	2905	100,0	100,0	287	1883	206	1	0:00:57
eil101	7	38,6	2926	99,9	100,1	1656	8074	34	9	0:06:53
eil101	9	18,8	1955	100,0	100,0	3411	4801	0	1	0:13:41
lin105	3	100,0	43137	100,0	100,0	56	821	92	1	0:00:38
lin105	5	80,0	69365	100,0	100,0	264	2018	355	1	0:01:08
lin105	7	53,3	83597	100,0	100,0	1413	5919	11	1	0:03:47
lin105	9	31,4	69920	100,0	100,0	4189	5320	5	1	0:13:05
pr107	3	100,0	132909	100,0	100,0	0	1142	64	1	0:00:42
pr107	5	68,2	210465	100,0	100,0	354	2866	238	1	0:01:16
pr107	7	43,0	259571	100,0	100,0	1400	6357	5	1	0:04:07
pr107	9	26,2	264918	100,0	100,0	5362	6317	0	1	0:16:06
gr120	3	100,0	20826	99,9	103,3	50	1405	180	11	0:02:49
gr120	5	75,8	31480	100,0	100,0	396	5684	276	1	0:03:47
gr120	7	41,7	32301	100,0	100,0	1210	6202	0	1	0:06:38
gr120	9	22,5	24322	100,0	100,0	4411	6967	0	1	0:25:09

Table 7.2: Branch-and-Cut results for MCP1 on TSPLIB instances

name	α	p^*	opt	%-LB	%-UB	pair	gsec	2mat	nodes	time
pr124	3	100,0	177090	98,8	105,0	137	4788	964	69	0:10:08
pr124	5	90,3	286115	99,8	103,5	609	10979	1126	13	0:08:59
pr124	7	66,9	358853	100,0	100,0	1237	5712	19	1	0:05:17
pr124	9	39,5	340153	100,0	100,0	5135	9247	0	1	0:34:01
bier127	3	100,0	354846	99,8	104,7	58	980	313	13	0:03:16
bier127	5	76,4	539955	100,0	108,2	527	10544	841	3	0:07:05
bier127	7	44,1	567110	100,0	101,4	2529	12372	6	3	0:15:07
bier127	9	15,0	347845	100,0	100,0	5796	7519	0	1	1:12:15
ch130	3	100,0	18330	99,8	107,1	114	3074	411	33	0:06:20
ch130	5	83,8	28790	100,0	106,6	336	6333	584	3	0:05:32
ch130	7	47,7	32707	99,6	103,9	2748	17112	128	13	0:21:27
ch130	9	16,2	23639	100,0	100,0	5582	7371	0	1	0:43:40
pr136	3	100,0	290316	99,5	104,1	52	13931	4937	129	0:32:49
pr136	5	86,8	468520	100,0	100,0	442	8013	328	1	0:06:48
pr136	7	43,4	491981	100,0	100,0	1737	7941	0	1	0:10:08
pr136	9	25,7	387327	100,0	100,0	5979	9349	0	1	0:51:10
gr137	3	100,0	208929	99,8	104,1	66	2340	507	21	0:07:06
gr137	5	83,9	329465	99,7	101,2	618	20213	1525	11	0:17:25
gr137	7	54,7	366022	100,0	100,0	1866	10941	13	1	0:13:52
gr137	9	26,3	335009	100,0	100,0	6105	10369	0	1	1:01:24
pr144	3	100,0	175611	99,8	103,1	128	1913	397	13	0:04:19
pr144	5	94,4	290945	99,5	104,9	868	36880	4024	259	1:25:37
pr144	7	63,9	383041	100,0	100,0	2004	20733	211	1	0:21:12
pr144	9	23,6	366833	100,0	100,0	8182	10488	0	1	0:51:08
ch150	3	100,0	19584	99,8	104,4	58	3073	625	23	0:08:37
ch150	5	77,3	31170	100,0	100,0	537	11555	579	3	0:12:26
ch150	7	48,0	34930	100,0	104,0	1740	9981	2	3	0:18:12
ch150	9	18,0	26371	100,0	100,0	7106	10693	0	1	1:39:49
kroA150	3	100,0	79572	99,5	103,5	112	11424	2068	103	0:31:50
kroA150	5	80,7	125435	100,0	100,0	479	8117	463	1	0:08:22
kroA150	7	48,7	140961	99,8	104,0	2244	12360	81	13	0:23:21
kroA150	9	19,3	113080	100,0	100,0	6906	9767	0	1	1:27:46
kroB150	3	100,0	78390	99,5	107,3	182	8483	2641	135	0:41:23
kroB150	5	77,3	128425	95,2	100,0	2474	100970	3921	111	>2:00:00
kroB150	7	50,0	135382	100,0	100,0	1690	7891	4	1	0:13:10
kroB150	9	18,7	108885	100,0	100,0	6985	10424	0	1	1:28:37
pr152	3	100,0	221046	99,5	103,4	130	3514	1383	65	0:22:20
pr152	5	87,5	376155	96,0	100,0	1595	58290	6674	285	>2:00:00
pr152	7	51,3	475052	96,7	100,0	5576	84188	2042	59	>2:00:00
pr152	9	21,1	475440	100,0	100,0	10728	14522	5	1	1:30:10
u159	3	100,0	126240	99,8	107,8	58	1570	428	13	0:06:13
u159	5	86,8	204250	100,0	104,4	818	18906	1575	11	0:23:17
u159	7	56,0	235221	100,0	100,0	1890	9551	8	1	0:16:05
u159	9	25,8	199552	100,0	100,0	7951	13408	0	1	1:51:41
si175	3	100,0	64557	99,4	100,0	0	28547	7573	283	>2:00:00
si175	5	81,7	108060	98,2	100,0	1964	107862	3479	1	>2:00:00
si175	7	11,4	90560	98,2	100,0	6327	17235	0	1	>2:00:00
si175	9	2,9	43025	96,1	100,0	14778	11629	0	1	>2:00:00
rat195	3	100,0	6969	99,7	108,4	98	23958	1868	73	1:24:24
rat195	5	85,1	11320	99,9	108,3	695	16474	1058	13	0:34:14
rat195	7	40,0	12319	100,0	100,0	2914	23126	15	1	1:14:16
rat195	9	16,9	9395	88,6	100,0	7396	13965	0	1	>2:00:00
d198	3	100,0	47340	99,7	102,5	72	12461	1278	121	1:17:37
d198	5	81,8	77075	99,8	102,3	2518	85335	5214	3	>2:00:00
d198	7	48,5	94541	99,6	101,1	5342	46508	266	11	>2:00:00
d198	9	19,2	97899	89,2	100,0	13038	21519	0	1	>2:00:00
kroA200	3	100,0	93699	93,6	100,0	234	23765	2930	129	>2:00:00
kroA200	5	69,5	149685	92,6	100,0	1339	63010	2704	9	>2:00:00
kroA200	7	48,0	164194	96,2	100,0	4461	38866	372	13	>2:00:00
kroA200	9	19,0	124678	90,7	100,0	7529	10163	0	1	>2:00:00
kroB200	3	100,0	88311	99,8	106,3	80	2827	852	43	0:26:57
kroB200	5	77,0	138905	99,9	104,7	1098	25428	1007	7	0:49:08
kroB200	7	49,0	156638	100,0	100,0	3006	17594	3	1	0:54:44
kroB200	9	18,5	127800	89,6	100,0	7541	10243	0	1	>2:00:00

Table 7.3: Branch-and-Cut results for MCP1 on TSPLIB instances

name	β	p^*	opt	%-LB	%-UB	pair	gsec	2mat	cover	sec2	cc	nodes	time
burma14	0	100,0	3323	100,0	100,0	0	0	0	0	3	0	1	0:00:00
burma14	0,08	85,7	3114	100,0	100,0	15	0	1	4	52	0	1	0:00:00
burma14	0,22	78,6	2592	99,1	105,8	16	0	2	7	58	3	9	0:00:01
burma14	0,42	71,4	1967	98,6	101,2	24	9	1	9	125	31	7	0:00:01
ulysses16	0	100,0	6747	100,0	100,0	0	0	3	0	8	0	1	0:00:00
ulysses16	0,08	81,3	6413	99,4	101,3	15	0	3	9	52	5	13	0:00:01
ulysses16	0,22	68,8	5944	99,7	105,7	28	4	1	3	84	14	5	0:00:01
ulysses16	0,42	75,0	4371	100,0	100,0	25	5	5	19	41	3	1	0:00:01
gr17	0	100,0	2085	100,0	100,0	0	0	0	0	13	0	1	0:00:00
gr17	0,08	76,5	1932	99,9	100,0	22	0	0	4	28	7	3	0:00:00
gr17	0,22	88,2	1476	98,9	100,0	21	10	0	16	42	3	9	0:00:01
gr17	0,42	58,8	1083	99,6	103,3	38	29	0	3	93	21	5	0:00:01
gr21	0	100,0	2707	100,0	100,0	0	0	0	0	0	0	1	0:00:00
gr21	0,08	85,7	2404	99,1	100,4	37	0	0	14	261	8	11	0:00:02
gr21	0,22	76,2	2006	99,4	102,8	43	34	2	8	321	8	5	0:00:02
gr21	0,42	66,7	1599	98,6	101,4	46	30	8	8	246	23	7	0:00:02
ulysses22	0	100,0	6901	100,0	100,0	0	0	0	0	17	0	1	0:00:00
ulysses22	0,08	72,7	6476	99,1	102,7	34	6	2	19	197	22	33	0:00:05
ulysses22	0,22	63,6	5820	99,5	100,8	54	39	9	19	518	42	19	0:00:05
ulysses22	0,42	81,8	3657	99,4	100,0	38	39	1	5	257	27	5	0:00:02
gr24	0	100,0	1272	100,0	100,0	0	0	3	0	4	0	1	0:00:00
gr24	0,08	87,5	1088	98,7	100,0	43	1	13	48	334	14	47	0:00:07
gr24	0,22	66,7	840	99,0	100,5	50	24	7	4	583	34	21	0:00:05
gr24	0,42	41,7	647	99,5	100,8	78	94	1	2	391	24	9	0:00:04
fri26	0	100,0	937	100,0	100,0	0	0	7	0	22	0	1	0:00:00
fri26	0,08	88,5	804	99,4	100,0	36	4	26	28	283	7	23	0:00:05
fri26	0,22	53,8	658	98,9	101,7	66	95	18	1	440	28	15	0:00:06
fri26	0,42	53,8	530	98,5	102,5	97	330	0	7	932	78	15	0:00:09
bayg29	0	100,0	1610	100,0	100,0	0	0	7	0	10	0	1	0:00:01
bayg29	0,08	89,7	1378	99,7	104,2	26	1	25	7	109	3	7	0:00:03
bayg29	0,22	72,4	1021	99,4	100,0	56	35	10	14	982	45	17	0:00:08
bayg29	0,42	48,3	743	98,8	105,4	89	174	1	9	694	68	27	0:00:10
bays29	0	100,0	2020	100,0	100,0	0	0	23	0	17	0	1	0:00:01
bays29	0,08	89,7	1703	99,1	103,3	42	7	54	41	367	43	31	0:00:08
bays29	0,22	69,0	1224	97,8	100,2	81	127	21	37	1954	173	49	0:00:17
bays29	0,42	48,3	846	98,2	101,4	87	138	0	4	724	83	13	0:00:07
dantzig42	0	100,0	699	99,9	102,4	0	0	19	0	13	0	5	0:00:02
dantzig42	0,08	81,0	571	99,5	109,6	70	25	44	15	849	121	13	0:00:17
dantzig42	0,22	54,8	437	99,1	103,2	125	215	7	2	1447	176	13	0:00:21
dantzig42	0,42	38,1	321	99,4	100,6	185	381	5	0	798	64	7	0:00:19
swiss42	0	100,0	1273	100,0	100,0	0	0	9	0	51	0	1	0:00:02
swiss42	0,08	85,7	1039	100,0	100,4	48	19	3	23	349	18	7	0:00:09
swiss42	0,22	61,9	750	99,5	100,1	119	305	20	6	1146	53	21	0:00:23
swiss42	0,42	42,9	564	98,9	105,1	200	903	0	8	1371	137	25	0:00:39
att48	0	100,0	10628	99,8	101,9	0	0	84	0	151	0	5	0:00:07
att48	0,08	83,3	8574	99,7	100,5	90	134	77	53	2151	93	43	0:00:52
att48	0,22	62,5	6257	99,4	103,9	162	518	31	8	2968	101	15	0:00:43
att48	0,42	39,6	4654	99,2	100,8	303	1073	0	9	2512	137	33	0:01:26
gr48	0	100,0	5046	99,7	102,8	0	0	135	0	158	0	11	0:00:10
gr48	0,08	85,4	3995	99,8	101,5	54	33	59	13	560	32	13	0:00:21
gr48	0,22	64,6	2673	99,6	101,3	120	231	21	9	1760	131	15	0:00:30
gr48	0,42	45,8	1772	99,9	100,0	183	256	3	3	547	24	3	0:00:15
hk48	0	100,0	11461	100,0	100,0	0	0	3	0	60	0	1	0:00:03
hk48	0,08	83,3	9425	99,9	100,4	77	56	12	19	915	44	21	0:00:28
hk48	0,22	56,3	6971	99,6	100,0	156	441	26	6	1900	102	41	0:00:59
hk48	0,42	37,5	4734	99,8	101,2	212	658	0	4	1261	27	13	0:00:35
eil51	0	100,0	426	100,0	104,7	0	0	82	0	64	0	3	0:00:04
eil51	0,08	86,3	346	99,7	102,6	82	76	189	25	1813	114	37	0:00:39
eil51	0,22	60,8	246	100,0	102,8	130	572	32	0	904	29	3	0:00:19
eil51	0,42	35,3	171	98,8	100,6	255	1150	7	4	1086	84	29	0:01:11
berlin52	0	100,0	7542	100,0	100,0	0	0	1	0	6	0	1	0:00:02
berlin52	0,08	82,7	6292	99,9	100,4	117	122	27	50	2224	80	23	0:00:39
berlin52	0,22	63,5	4665	99,3	101,5	137	1165	114	40	4085	209	47	0:01:36
berlin52	0,42	50,0	3143	99,7	100,2	203	666	6	8	979	36	15	0:00:39

Table 7.4: Branch-and-Cut results for MCP2 of TSPLIB instances

name	β	p^*	opt	%-LB	%-UB	pair	gsec	2mat	cover	sec2	cc	nodes	time
brazil58	0	100,0	25395	100,0	100,0	0	0	3	0	26	0	1	0:00:05
brazil58	0,08	79,3	19861	99,9	100,5	74	191	73	13	841	90	9	0:00:39
brazil58	0,22	55,2	14548	100,0	100,2	174	904	26	5	2062	119	9	0:00:56
brazil58	0,42	44,8	10497	99,8	100,4	286	1715	13	7	2361	48	27	0:01:56
st70	0	100,0	675	99,9	103,1	0	0	66	0	70	0	5	0:00:11
st70	0,08	77,1	495	99,8	103,4	122	237	235	14	2413	144	17	0:01:17
st70	0,22	47,1	339	100,0	101,2	237	860	14	2	1389	72	3	0:00:56
st70	0,42	28,6	239	100,0	102,1	417	1221	0	1	1254	23	13	0:02:20
eil76	0	100,0	538	100,0	100,0	0	0	45	0	28	0	1	0:00:09
eil76	0,08	85,5	407	99,8	104,4	138	344	277	56	5790	173	41	0:02:59
eil76	0,22	55,3	267	99,3	102,2	224	2019	68	0	1761	22	7	0:01:51
eil76	0,42	32,9	164	100,0	101,8	427	2258	0	0	1636	19	11	0:03:10
pr76	0	100,0	108159	98,6	103,4	0	0	3370	0	9166	0	425	0:17:12
pr76	0,08	76,3	79856	99,1	107,2	330	8646	4408	72	48845	1257	625	0:50:15
pr76	0,22	51,3	55212	98,6	107,7	1025	52797	6317	12	55950	868	851	1:33:17
pr76	0,42	36,8	36885	99,7	100,0	531	2769	2	17	3479	75	57	0:08:22
gr96	0	100,0	54645	99,3	104,8	0	0	430	0	972	0	43	0:05:33
gr96	0,08	81,3	37227	99,9	100,5	183	1007	292	27	8206	232	35	0:12:58
gr96	0,22	47,9	23435	99,9	100,8	388	1788	4	3	3760	163	23	0:09:04
gr96	0,42	21,9	18116	99,7	100,3	933	2950	0	8	2649	144	37	0:28:34
rat99	0	100,0	1211	99,9	103,5	0	0	111	0	581	0	7	0:01:01
rat99	0,08	75,8	911	99,8	100,1	309	3008	322	27	18925	413	111	0:21:57
rat99	0,22	43,4	593	99,8	101,5	399	2555	4	2	3834	75	17	0:06:31
rat99	0,42	22,2	414	99,8	102,9	841	3214	0	1	2368	202	7	0:14:41
kroA100	0	100,0	21282	99,8	100,8	0	0	204	0	361	0	15	0:01:22
kroA100	0,08	68,0	14781	99,7	101,1	249	2176	259	41	19410	1012	79	0:19:33
kroA100	0,22	38,0	10474	99,7	101,6	567	4580	0	11	5027	338	75	0:22:24
kroA100	0,42	25,0	7416	99,7	103,8	895	3404	0	4	1558	21	23	0:19:19
kroB100	0	100,0	22141	99,5	104,0	0	0	481	0	1414	0	35	0:03:30
kroB100	0,08	73,0	15828	99,9	100,6	201	2042	289	79	13701	372	89	0:17:59
kroB100	0,22	40,0	10910	99,9	100,6	461	3156	13	18	5465	87	59	0:14:00
kroB100	0,42	26,0	7798	99,7	100,2	780	2939	0	3	2467	59	19	0:15:52
kroC100	0	100,0	20749	99,8	102,8	0	0	238	0	371	0	9	0:01:06
kroC100	0,08	69,0	14953	99,9	100,2	273	2794	319	73	21834	917	81	0:22:23
kroC100	0,22	39,0	10180	99,9	100,3	445	1940	0	3	3034	29	23	0:08:58
kroC100	0,42	24,0	7369	99,5	100,5	892	3789	0	4	2529	115	29	0:18:43
kroD100	0	100,0	21294	99,6	103,5	0	0	244	0	755	0	11	0:01:26
kroD100	0,08	68,0	15331	99,8	101,5	285	3654	227	85	18175	449	47	0:16:59
kroD100	0,22	40,0	10517	99,9	100,7	446	2652	9	5	3374	220	35	0:10:24
kroD100	0,42	27,0	7448	99,5	102,8	916	4188	0	4	2828	370	43	0:27:05
kroE100	0	100,0	22068	99,3	103,8	0	0	697	0	1067	0	35	0:03:33
kroE100	0,08	70,0	15788	99,8	102,8	267	5650	787	20	16550	620	79	0:19:39
kroE100	0,22	45,0	10726	99,9	102,9	451	4311	15	10	6262	142	51	0:17:40
kroE100	0,42	27,0	7682	99,9	100,0	807	2125	0	5	1954	48	17	0:15:58
rd100	0	100,0	7910	100,0	100,0	0	0	82	0	119	0	1	0:00:28
rd100	0,08	69,0	5692	99,8	103,3	215	4319	369	14	13923	154	45	0:15:24
rd100	0,22	41,0	3561	99,7	100,1	610	5705	45	26	12712	210	115	0:26:40
rd100	0,42	25,0	2424	99,9	101,2	841	2514	0	2	2209	53	9	0:16:56
eil101	0	100,0	629	99,8	105,7	0	0	97	0	345	0	5	0:00:49
eil101	0,08	81,2	463	100,0	100,2	415	243	110	83	6962	170	7	0:05:05
eil101	0,22	53,5	287	99,7	108,7	1859	8299	260	0	6194	44	19	0:11:25
eil101	0,42	25,7	167	100,0	102,4	2439	5424	0	1	1721	32	15	0:13:46
lin105	0	100,0	14379	100,0	100,0	0	0	96	0	45	0	1	0:00:32
lin105	0,08	64,8	10336	99,9	100,4	4623	12308	687	171	138132	991	471	>2:00:00
lin105	0,22	41,0	7937	99,6	102,3	4182	8309	7	3	10354	210	63	0:28:24
lin105	0,42	33,3	5549	99,9	100,7	3500	3784	10	7	4160	85	25	0:17:40
pr107	0	100,0	44303	100,0	100,0	0	0	11	0	209	0	1	0:00:40
pr107	0,08	48,6	31550	99,8	100,1	6617	12835	98	231	65499	7392	747	>2:00:00
pr107	0,22	25,2	25451	99,9	100,3	3623	4194	0	2	3741	54	9	0:21:40
pr107	0,42	18,7	22847	99,6	101,1	14721	8568	2	8	8717	3035	47	1:21:46
gr120	0	100,0	6942	99,9	105,9	0	0	244	0	759	0	11	0:02:58
gr120	0,08	75,0	4666	100,0	107,2	633	2517	510	8	9259	102	13	0:13:40
gr120	0,22	42,5	2692	100,0	100,6	1473	4050	5	3	4891	53	7	0:12:09
gr120	0,42	21,7	1755	99,9	100,9	3799	5487	1	3	3839	38	15	0:37:14

Table 7.5: Branch-and-Cut results for MCP2 of TSPLIB instances

name	β	p^*	opt	%-LB	%-UB	pair	gsec	2mat	cover	sec2	cc	nodes	time
pr124	0	100,0	59030	98,8	104,4	0	0	979	0	2332	0	57	0:11:02
pr124	0,08	62,9	41275	100,0	103,2	1139	2129	92	0	5184	150	3	0:09:25
pr124	0,22	42,7	31173	99,8	101,4	4023	5431	3	33	8657	73	39	0:40:22
pr124	0,42	29,8	25085	99,7	100,9	9186	8673	0	7	7093	404	11	>2:00:00
bier127	0	100,0	118282	99,8	108,5	0	0	185	0	344	0	9	0:02:04
bier127	0,08	81,9	83355	99,8	104,8	1684	5720	2234	295	56576	1025	189	1:26:37
bier127	0,22	46,5	52212	100,0	100,1	2114	9791	93	3	8338	97	45	0:30:28
bier127	0,42	33,1	29061	99,9	101,3	6285	13008	55	44	15177	196	151	1:34:15
ch130	0	100,0	6110	99,9	107,6	0	0	271	0	1122	0	25	0:04:24
ch130	0,08	69,2	4225	99,9	103,3	1154	6775	1023	9	18704	337	31	0:30:51
ch130	0,22	40,0	2773	99,5	107,0	4546	11863	73	12	13249	247	63	0:49:46
ch130	0,42	19,2	1776	99,7	108,1	7972	9296	0	9	6252	119	59	1:24:54
pr136	0	100,0	96772	99,5	105,3	0	0	9056	0	20561	0	281	1:14:30
pr136	0,08	70,6	67294	99,8	102,2	3064	14827	1544	187	57085	6832	141	>2:00:00
pr136	0,22	34,6	39310	99,9	100,2	4358	4858	0	3	4756	186	47	0:45:57
pr136	0,42	23,5	27779	99,6	101,4	8804	9463	0	2	5173	36	9	1:57:07
gr137	0	100,0	69643	99,8	108,3	0	0	303	0	516	0	27	0:09:43
gr137	0,08	71,5	45250	100,0	100,0	1058	2090	731	3	9199	298	15	0:35:09
gr137	0,22	33,6	31382	99,9	100,5	5019	7038	1	21	8031	299	67	1:18:33
gr137	0,42	20,4	25356	99,9	100,6	9408	5868	0	14	6476	124	31	>2:00:00
pr144	0	100,0	58537	99,8	100,9	0	0	261	0	858	0	9	0:04:23
pr144	0,08	50,0	43093	99,8	101,2	4383	13630	99	4	22032	5252	19	>2:00:00
pr144	0,22	33,3	32288	99,9	100,2	6285	7451	4	30	18773	237	77	1:45:03
pr144	0,42	28,5	25610	95,8	100,0	9532	7795	2	8	4979	64	1	>2:00:00
ch150	0	100,0	6528	99,8	104,7	0	0	625	0	2119	0	25	0:10:04
ch150	0,08	66,0	4364	99,9	100,6	1763	13712	483	25	20899	202	23	0:50:54
ch150	0,22	39,3	2808	99,8	103,1	4641	12553	67	12	10747	137	59	1:20:33
ch150	0,42	25,3	1990	89,7	100,0	5677	5925	0	2	3388	25	1	>2:00:00
kroA150	0	100,0	26524	99,5	106,8	0	0	1760	0	9176	0	103	0:34:15
kroA150	0,08	70,7	17718	98,3	101,6	4237	27481	2625	3	60239	294	79	>2:00:00
kroA150	0,22	36,0	11363	99,9	100,4	4614	10820	3	18	6974	225	99	1:41:36
kroA150	0,42	26,0	8370	92,5	100,0	6306	7875	0	4	2601	13	1	>2:00:00
kroB150	0	100,0	26130	99,5	106,3	0	0	2367	0	7962	0	105	0:45:17
kroB150	0,08	66,7	16875	99,9	107,1	1504	10340	922	25	23769	534	57	1:02:26
kroB150	0,22	35,3	10568	99,9	101,0	3318	7248	3	14	6886	95	45	1:03:36
kroB150	0,42	19,3	7633	96,8	100,0	6806	6855	0	4	3353	11	1	>2:02:00
pr152	0	100,0	73682	99,5	101,9	0	0	881	0	2040	0	35	0:16:16
pr152	0,08	27,6	53670	95,6	100,0	6829	19054	1282	0	15318	194	43	>2:00:00
pr152	0,22	27,6	43956	93,3	100,0	9272	7807	1	5	5446	78	1	>2:00:00
pr152	0,42	5,9	36830	84,9	100,0	12496	5290	0	9	3865	83	1	>2:00:00
u159	0	100,0	42080	99,8	107,9	0	0	688	0	1575	0	13	0:07:33
u159	0,08	64,8	27635	99,8	108,3	2282	9032	622	8	36757	350	57	1:32:01
u159	0,22	37,1	17666	100,0	102,7	4199	7827	0	4	6586	105	21	1:26:04
u159	0,42	22,0	13579	92,6	100,0	8486	5322	0	4	3543	92	1	>2:00:00
si175	0	100,0	21463	99,6	100,7	0	0	5819	0	13923	0	247	>2:00:00
si175	0,08	88,0	18799	97,8	100,0	1378	6042	2501	9	37200	142	43	>2:00:00
si175	0,22	61,7	13875	97,5	100,0	2773	32820	1226	0	31996	32	1	>2:00:00
si175	0,42	29,7	7548	99,4	102,4	4936	20323	19	13	7456	964	69	>2:00:00
rat195	0	100,0	2323	99,7	107,3	0	0	1426	0	15543	0	53	1:13:05
rat195	0,08	67,7	1663	93,2	100,0	1616	28563	476	0	23135	103	9	>2:00:00
rat195	0,22	34,9	869	99,8	100,0	3594	14088	1	0	7925	6	7	>2:00:00
rat195	0,42	13,3	602	73,4	100,0	8090	5480	0	5	6167	20	1	>2:00:00
d198	0	100,0	15780	99,7	102,8	0	0	799	0	4803	0	81	0:57:32
d198	0,08	50,0	11092	98,4	100,0	5095	11510	186	0	19454	269	1	>2:00:00
d198	0,22	23,7	9055	88,4	100,0	10892	3790	0	7	6012	314	1	>2:00:00
d198	0,42	4,0	8265	69,2	100,0	10783	2219	0	18	3398	293	1	>2:00:00
kroA200	0	100,0	29909	97,7	103,5	0	0	3380	0	21215	0	105	>2:00:00
kroA200	0,08	59,0	18394	99,4	109,1	2560	16805	495	0	24195	133	13	>2:00:00
kroA200	0,22	20,0	12964	86,2	100,0	4756	11858	2	2	7147	34	1	>2:00:00
kroA200	0,42	10,0	8017	70,4	100,0	7810	4783	0	8	4373	31	1	>2:00:00
kroB200	0	100,0	29437	99,8	107,5	0	0	1008	0	3912	0	37	0:33:54
kroB200	0,08	61,5	18296	99,9	100,1	2652	17265	464	11	23208	177	19	>2:00:00
kroB200	0,22	20,5	12216	93,2	100,0	4610	8788	0	1	5733	60	1	>2:00:00
kroB200	0,42	15,5	8035	75,3	100,0	7527	4366	0	4	3422	21	1	>2:00:00

Table 7.6: Branch-and-Cut results for MCP2 of TSPLIB instances

$\|V\|$	α	succ	p^*	%-LB	%-UB	pair	gsec	2mat	nodes	time
25	3	10	100,0	99,7	101,1	9,0	75,6	12,2	2,4	0:00:00
	5	10	76,8	100,0	100,2	34,4	166,3	9,0	1,2	0:00:00
	7	10	43,2	100,0	100,0	86,4	314,9	0,1	1,0	0:00:01
	9	10	12,0	100,0	100,0	143,4	161,0	0,0	1,0	0:00:01
50	3	10	100,0	99,7	103,1	16,3	422,4	85,3	11,2	0:00:07
	5	10	79,2	99,9	100,1	69,6	919,6	59,2	3,6	0:00:07
	7	10	49,6	100,0	100,0	199,3	938,0	1,1	1,0	0:00:07
	9	10	12,4	100,0	100,0	549,2	1004,2	0,0	1,0	0:00:21
75	3	10	100,0	99,8	102,7	21,6	547,5	124,1	7,0	0:00:18
	5	10	78,5	99,9	100,7	114,0	3537,0	314,6	17,8	0:01:01
	7	10	49,2	100,0	100,3	321,8	2236,3	6,2	1,6	0:00:44
	9	10	15,7	100,0	100,0	891,5	2376,1	0,0	1,0	0:03:20
100	3	10	100,0	99,8	104,1	35,0	1561,1	388,8	22,0	0:01:49
	5	10	77,9	99,9	102,3	151,1	5469,4	448,7	7,0	0:02:25
	7	10	46,0	99,8	100,9	481,3	13160,5	421,6	43,6	0:11:50
	9	10	19,4	100,0	100,0	1267,4	4069,0	0,0	1,0	0:13:13
125	3	10	100,0	99,7	105,1	58,9	3404,4	493,9	28,4	0:05:22
	5	10	80,0	99,9	103,7	502,1	11587,1	750,7	12,6	0:08:31
	7	10	45,4	99,9	100,5	2008,2	11912,3	141,8	10,0	0:12:13
	9	10	18,2	100,0	100,0	5278,5	7530,9	0,0	1,0	0:40:56
150	3	9	100,0	99,8	104,9	123,7	8895,9	1426,6	83,7	0:26:10
	5	10	80,3	99,9	103,0	944,2	34802,7	1632,8	28,4	0:34:44
	7	9	47,0	100,0	101,3	2206,6	13492,7	70,1	4,3	0:20:33
	9	7	19,2	100,0	100,0	6949,1	10346,6	0,0	1,0	1:32:27
175	3	10	100	99,8	106,0	86,8	3900,2	788,4	29,0	0:15:34
	5	6	80,2	99,9	103,0	809,3	27332,5	1409,8	10,7	0:34:51
	7	7	46,1	99,9	101,3	3240,4	20893,0	258,4	8,7	0:44:09
	9	0	**	**	**	**	**	**	**	**
200	3	8	100,0	99,8	106,8	128,9	6277,6	1326,0	48,0	0:38:30
	5	9	79,6	99,9	103,9	1099,3	44796,1	2137,7	12,1	1:12:22
	7	5	48,3	100,0	100,9	3141,2	20909,2	12,0	3,0	0:54:34
	9	0	**	**	**	**	**	**	**	**

Table 7.7: Branch-and-Cut results for MCP1 on random instances

$\|V\|$	β	succ	p^*	%-LB	%-UB	pair	gsec	2mat	cover	sec2	cc	nodes	time
25	0	10	100,0	99,8	100,8	0,0	0,0	11,2	0,0	35,6	0,0	2,0	0:00:00
	0,08	10	84,8	99,6	101,1	34,2	2,0	11,6	14,7	252,9	13,2	9,4	0:00:02
	0,22	10	69,2	99,4	100,8	48,1	30,2	4,6	11,6	357,1	37,6	12,8	0:00:03
	0,42	10	48,4	99,4	101,1	78,0	117,2	3,0	4,7	499,1	46,6	11,8	0:00:04
50	0	10	100,0	99,7	101,3	0,0	0,0	84,4	0,0	172,2	0,0	9,2	0:00:07
	0,08	10	81,0	99,6	101,7	107,0	440,4	515,6	68,1	3990,5	187,4	116,0	0:02:26
	0,22	10	55,2	99,7	101,0	157,5	592,4	15,5	7,9	1826,6	142,8	20,2	0:00:44
	0,42	10	36,0	99,4	103,2	273,5	849,5	0,6	6,1	1264,0	68,0	30,2	0:01:12
75	0	10	100,0	99,8	101,6	0,0	0,0	115,3	0,0	325,8	0,0	8,8	0:00:24
	0,08	10	74,8	99,7	102,4	184,7	1586,5	405,7	71,2	11780,9	366,9	88,6	0:07:14
	0,22	10	47,3	99,7	101,4	308,8	2190,2	25,0	6,5	3442,0	139,7	36,0	0:04:07
	0,42	10	28,9	99,6	103,2	537,5	2215,4	0,1	4,0	1958,7	0,0	26,2	0:06:00
100	0	10	100,0	99,7	102,8	0,0	0,0	377,4	0,0	992,7	0,0	20,8	0:02:05
	0,08	10	71,5	99,7	103,9	282,2	6830,5	714,8	43,0	24663,8	396,5	100,8	0:27:10
	0,22	10	41,8	99,6	102,7	557,7	9765,4	325,6	5,2	8560,2	218,6	73,8	0:24:48
	0,42	10	24,0	99,9	102,9	887,2	3227,2	0,0	2,0	1998,3	43,8	22,4	0:20:14
125	0	10	100,0	99,8	105,1	0,0	0,0	489,0	0,0	2573,4	0,0	30,6	0:07:14
	0,08	8	69,2	99,8	102,5	1532,9	9346,5	713,8	21,8	24680,6	291,9	47,3	0:37:51
	0,22	9	40,0	99,7	102,1	3344,8	10085,1	126,4	8,6	9479,7	133,3	44,6	0:35:17
	0,42	10	21,8	99,9	103,3	5143,3	6689,5	0,0	6,0	3539,2	64,1	25,4	0:55:06
150	0	8	100,0	99,8	105,1	0,0	0,0	1044,5	0,0	5090,6	0,0	47,5	0:19:44
	0,08	6	66,0	99,9	104,4	1849,5	15577,8	955,5	13,2	29003,3	238,8	32,0	1:05:53
	0,22	7	36,7	99,9	101,2	3680,0	12140,9	25,6	6,7	6571,4	125,4	30,7	0:58:54
	0,42	1	17,3	99,9	100,8	7388,0	8415,0	0,0	3,0	5121,0	156,0	9,0	1:55:30
175	0	10	100,0	99,8	104,3	0,0	0,0	660,9	0,0	2296,5	0,0	25,2	0:16:00
	0,08	3	64,2	100,0	103,0	1869,0	15626,3	743,3	17,7	31571,7	337,7	20,3	1:32:33
	0,22	4	34,0	100,0	100,4	4730,8	14883,5	7,0	2,3	7013,8	77,8	20,0	1:34:58
	0,42	0	**	**	**	**	**	**	**	**	**	**	**
200	0	7	100,0	99,8	106,1	0,0	0,0	1143,1	0,0	4134,6	0,0	37,0	0:36:47
	0,08	1	62,0	99,8	106,6	2186,0	21390,0	929,0	5,0	18274,0	141,0	23,0	1:44:58
	0,22	0	**	**	**	**	**	**	**	**	**	**	**
	0,42	0	**	**	**	**	**	**	**	**	**	**	**

Table 7.8: Branch-and-Cut results for MCP2 on random instances

7.2 Heuristic results

Eliminating the shaking step from the VNTS algorithm described in Chapter 6, a Tabu Search algorithm for MCP is obtained. If, in addition, the tabu list is also eliminated, the algorithm becomes a Local Search method. Restarting the Local Search a given number of times we have a Multistart solution method. In this section we present the results obtained by the VNTS algorithm, and compare it with those obtained by the Local Search , Tabu Search and Multistart methods just mentioned. Our intention is to show that the Tabu list and the shake procedure used in VNTS contribute to make the method more efficient.

We also compare VNTS with two other heuristics methods, not so related to the VNTS algorithm as the former three ones. The first one is the constructive method described in Chapter 5 to generate initial solutions at the beginning of the Branch-and-Cut algorithm. The second one is a pure Variable Neighbourhood Descent method (Mladenovic [53]), with three neighbourhood structures, those given by the add, drop and add-drop moves.

All the heuristic algorithms were implemented in C++ and tested on instances from the TSPLIB involving between 50 and 500 nodes and having EUC2D format (Euclidean distance). Tables 7.9 to 7.12 summarize the computational results for MCP1 and MCP2 using the different heuristic methods. The Multistart was performed 5 times and the results reported were taken in average. For instances with at most 200 nodes (Tables 7.9 and 7.11) we compare the heuristic results with those given by the Branch-and-Cut method. For instances with more than 200 nodes (Tables 7.10 and 7.12) we compare the values given by the different heuristic methods among them. In all cases, the best results for each instance are written in bold characters.

In Tables 7.10 and 7.12 the column heading are defined as follows:

Name: problem name.

$\boldsymbol{\alpha}$ / $\boldsymbol{\beta}$: scale factor described at the beginning of the chapter.

LAM: value given by the constructive method use at the beginning of the Branch-and-Cut algorithm described in Chapter 6.

VND: value given by the Variable Neighbourhood Descent heuristic.

VNTS: value given by the Variable Neighbourhood Tabu Search described in Chapter 7.

TS: value given by the Tabu Search.

LS: value given by the Local Search.

MS: value given by the MultiStart.

Columns **HEUR-t** show the CPU time, in seconds, spent by each heuristic method. In Table 7.9 and Table 7.11, the columns **HEUR-E** show the errors, computed as *100(bc - heur)/bc*, where *bc* the solution given by the Branch-and-Cut algorithm, and *heur* the heuristic solution. The solution given by the Branch-and-Cut algorithm is not really the optimal one in all the cases, since there was a time limit of two hours in the execution time for each instance. This explains the negative values that appear in the error columns: they correspond to instances in which the heuristic solution is better than the best solution found by the Branch-and-Cut algorithm in two hours.

Finally, Table 7.13 summarizes the information about VNTS given by the previous four tables. It shows the percentage of times that VNTS gives the best result when compared with the other heuristics. The first column of the table indicates the size of the solution (depending on the value of the scale factor). The other columns correspond to:

MCP1-200 MCP1 on instances with less at most 200 nodes.

MCP1-500 MCP1 on instances with a number of nodes between 200 and 500.

MCP2-200 MCP2 on instances with less at most 200 nodes.

MCP2-500 MCP2 on instances with a number of nodes between 200 and 500.

From Table 7.9 and Table 7.11 we can conclude that VNTS gives better results for MCP1 than for MCP2. Regarding the solution size, the performance of VNTS is better when the solution is a small cycle ($\alpha = 9$ and

$\beta = 0.42$). The error percentage are not as good for medium and big cycles. However, when compared with the other heuristics, VNTS is the method that gives the largest percentage of 'best result obtained' (bold characters). This is also true for Tables 7.10 and 7.12. As for the running time, it is acceptably small for instances with less than 200 nodes. It grows considerably for bigger instances, but the same happens to the other heuristic methods.

Name	α	LAM-E	LAMT-t	VNS-E	VNS-t	VNTS-E	VNTS-t	TS-E	TS-t	LS-E	LS-t	MS-E	MS-t
eil51	5	**2,76**	0,08	3,51	0,01	3,01	0,67	3,01	0,24	3,01	0,23	4,91	0,21
	7	1,47	0,02	2,27	0,01	**0,47**	0,58	1,99	0,14	1,99	0,14	1,28	0,14
	9	1,29	0,01	**0,00**	0,01	**0,00**	0,24	**0,00**	0,05	**0,00**	0,05	0,00	0,05
st70	5	2,41	0,25	2,25	0,05	1,13	2,10	1,45	0,85	1,45	0,85	**0,87**	0,80
	7	1,41	0,11	0,00	0,07	**0,00**	2,38	**0,00**	0,56	bf 0,00	0,59	0,09	0,55
	9	0,15	0,05	0,15	0,02	**0,00**	2,63	0,15	0,32	0,15	0,32	0,67	0,30
eil76	5	**0,61**	0,31	1,83	0,08	1,63	2,98	1,83	1,05	1,83	1,11	3,09	1,02
	7	2,76	0,13	1,88	0,04	4,19	3,82	1,80	0,64	1,80	0,64	**1,65**	0,65
	9	1,35	0,02	1,87	0,06	**0,29**	3,28	0,82	0,19	1,11	0,16	0,39	0,19
pr76	5	6,21	0,33	**3,41**	0,07	4,36	3,88	4,37	0,94	4,37	0,95	3,86	1,01
	7	3,89	0,18	1,00	0,04	**0,00**	3,43	1,00	0,74	1,00	0,74	0,93	0,77
	9	0,72	0,03	0,50	0,04	**0,18**	3,14	0,62	0,34	0,62	0,33	0,38	0,30
rat99	5	3,65	0,88	5,78	0,11	4,33	9,31	7,22	2,18	6,71	2,26	**3,60**	2,63
	7	4,26	0,40	2,86	0,15	**0,36**	7,05	2,63	1,65	2,63	1,59	1,17	1,62
	9	5,01	0,11	0,47	0,08	**0,27**	5,17	0,47	0,72	0,47	0,71	0,93	0,70
kroA100	5	**0,21**	0,96	0,44	0,10	0,44	12,42	0,44	3,05	0,44	3,02	2,78	2,88
	7	6,18	0,59	4,19	0,09	**0,00**	14,36	4,19	2,32	4,19	2,31	5,58	2,36
	9	3,15	0,08	0,49	0,12	0,39	2,69	**0,00**	0,77	**0,00**	0,78	1,74	0,62
kroB100	5	4,35	0,95	**1,05**	0,14	2,00	18,88	2,73	3,13	2,73	3,18	2,57	3,05
	7	3,95	0,35	0,57	0,12	**0,22**	6,85	0,57	1,83	0,57	1,90	1,99	1,76
	9	0,99	0,04	0,63	0,03	**0,09**	3,04	0,95	0,38	0,95	0,36	0,52	0,45
kroC100	5	3,43	1,00	3,26	0,11	**0,51**	12,73	2,40	3,11	2,40	3,07	3,05	3,06
	7	2,40	0,62	**0,00**	0,21	**0,00**	10,61	3,09	2,24	3,09	2,29	1,54	2,16
	9	2,07	0,14	0,45	0,09	**0,00**	4,85	**0,00**	0,86	bf 0,00	0,87	0,32	0,75
kroD100	5	5,25	0,93	**3,04**	0,13	3,09	6,36	3,09	2,28	3,04	2,55	4,60	2,93
	7	1,07	0,48	0,84	0,12	1,76	12,51	**0,69**	2,09	**0,69**	2,12	0,93	1,93
	9	2,82	0,10	1,87	0,05	**0,00**	6,19	0,56	0,74	0,56	0,74	0,12	0,79
kroE100	5	3,67	0,87	2,82	0,13	**0,59**	10,34	**0,59**	2,99	1,20	3,09	1,23	2,91
	7	5,60	0,47	0,46	0,24	0,75	10,95	**0,33**	2,21	**0,33**	2,27	0,82	2,16
	9	0,96	0,15	1,36	0,08	0,26	3,31	**0,01**	0,76	**0,01**	0,78	**0,01**	0,73
eil101	5	3,61	0,93	3,96	0,11	**2,41**	8,14	3,44	2,89	3,44	2,99	3,51	2,91
	7	2,29	0,37	1,06	0,11	**0,10**	5,05	0,21	1,56	0,21	1,57	1,58	1,58
	9	1,48	0,09	1,53	0,15	**0,00**	2,92	**0,00**	0,66	bf 0,00	0,65	0,15	0,60
pr107	5	0,89	1,07	**0,31**	0,12	**0,31**	15,12	**0,31**	3,64	**0,31**	3,60	1,37	3,49
	7	0,63	0,63	0,18	0,14	**0,00**	15,09	0,16	2,46	0,12	2,51	0,16	2,41
	9	1,08	0,17	0,68	0,08	**0,00**	10,56	0,60	1,00	0,60	1,00	0,38	1,01
pr136	5	1,43	3,57	**1,14**	0,39	3,47	29,14	3,73	11,44	3,73	11,68	2,84	11,25
	7	2,59	1,13	5,41	0,67	**0,15**	28,80	4,03	7,04	2,60	6,90	2,81	4,58
	9	4,14	0,47	1,02	0,39	0,19	16,25	0,30	2,34	0,30	2,36	**0,07**	2,69
pr144	5	4,87	4,47	2,28	0,26	**2,08**	35,71	**2,08**	9,61	**2,08**	10,48	2,77	11,33
	7	3,83	3,30	**0,51**	0,62	1,74	44,13	**0,51**	10,38	**0,51**	11,04	0,62	10,98
	9	4,68	0,61	0,01	0,76	**0,00**	18,24	0,16	3,06	0,16	2,96	0,06	3,20
kroA150	5	**2,62**	4,67	3,20	0,55	2,86	75,62	3,20	14,95	3,20	15,00	3,90	14,77
	7	3,97	2,31	0,98	0,30	1,51	38,21	**0,70**	9,42	**0,70**	9,40	1,87	9,60
	9	2,79	0,51	4,10	0,28	0,49	18,68	**0,06**	2,86	**0,06**	2,86	1,51	2,94
kroB150	5	0,00	4,85	-1,62	0,33	**-2,96**	28,70	**-2,96**	10,28	**-2,96**	11,42	-1,86	14,24
	7	4,48	2,20	1,21	0,93	1,66	32,19	**0,61**	9,42	**0,61**	9,82	1,55	9,15
	9	2,02	0,55	**0,07**	0,64	0,60	14,03	**0,07**	3,12	**0,07**	3,08	0,31	2,99
pr152	5	0,00	5,39	-0,86	0,44	**-2,12**	193,74	-0,13	15,24	-0,13	15,84	-0,37	13,95
	7	0,00	2,96	-0,57	0,69	-0,31	65,27	**-0,70**	11,61	**-0,70**	12,14	-0,60	11,42
	9	1,78	0,53	0,79	0,70	**0,39**	7,70	0,75	3,50	0,75	3,54	0,67	3,34
rat195	5	8,30	13,65	5,74	1,61	**5,65**	66,18	6,01	34,65	6,01	35,21	6,55	36,40
	7	2,53	5,05	4,81	3,16	**1,19**	65,61	2,07	22,28	2,07	22,72	1,96	19,35
	9	0,00	1,20	**-4,27**	1,04	-4,16	28,11	-4,19	6,30	-4,19	6,32	-3,84	6,48
kroA200	5	0,00	13,06	**-4,49**	1,92	-2,95	121,32	-2,88	43,99	-2,88	46,73	-2,96	44,89
	7	0,00	7,77	-0,65	1,46	-1,22	54,33	-0,42	23,80	-0,42	24,57	**-2,00**	29,01
	9	0,00	1,62	0,26	1,25	**-1,65**	32,75	-1,40	7,59	-1,40	7,64	-1,60	7,70
kroB200	5	4,97	14,22	4,93	1,58	**3,09**	71,93	**3,09**	41,89	**3,09**	43,97	4,45	44,21
	7	1,96	8,51	4,94	1,70	1,20	71,74	**0,71**	30,66	**0,71**	30,53	4,31	27,53
	9	0,00	1,54	1,43	1,26	-2,82	34,25	**-2,94**	7,12	**-2,94**	7,12	-2,49	6,53

Table 7.9: Heuristic results for MCP1 on instances with at most 200 nodes

Name	α	LAM	LAM-t	VNS	VNS-t	VNTS	VNTS-t	TS	TS-t	LS	LS-t	MS
ts225	5	644160	25,93	651695	2,99	642595	70,87	642595	53,45	643170	56,82	**642297,(**
	7	777025	19,81	764507	2,73	746586	169,63	763626	55,76	763626	58,43	**745870,(**
	9	678117	3,56	655569	1,18	**650769**	95,81	658011	13,57	658011	13,69	653910,6
tsp225	5	19870	21,52	20075	1,75	**19670**	158,27	19755	61,37	19760	68,15	19821,0
	7	21024	10,03	20744	6,85	20995	108,49	20761	32,89	20761	33,3	**20750,8**
	9	16466	2,83	15950	2,69	**15841**	135,28	15936	11,96	15936	12,02	15967,6
pr226	5	389690	25,28	393950	2,09	387410	151,65	387470	75,07	387470	75,04	**385731,(**
	7	477982	21,22	474744	2,36	470722	159,05	472607	62,54	472607	62,87	**469968,**
	9	499004	11,59	471748	1,35	**470711**	148,61	471748	40,18	471748	39,04	471332,0
pr264	5	252240	48,19	257410	2,47	254015	236,81	254215	163,42	254215	171,11	**249783,(**
	7	289968	28,64	287794	4,93	**287314**	164,1	287365	104,41	287365	100,68	287616,4
	9	257398	5,88	**256986**	2,67	257014	125,1	257014	25,98	257014	25,31	257064,4
a280	5	13475	61,14	13650	11,03	**13295**	535,31	13380	234,94	13380	238,13	13578,0
	7	**14341**	27,34	14701	5,59	14448	254,39	14390	129,17	14390	128,39	14718,4
	9	11155	6,71	10846	8,2	**10757**	172,96	10824	32,24	10824	30,97	10830,8
pr299	5	243220	79,42	244305	6,19	244170	508,74	244345	345,04	244345	348,12	**241036,(**
	7	269679	47,7	**264309**	7,97	265182	512,52	268423	207,45	268423	203,35	267095,0
	9	222242	13,21	214861	2,78	**213857**	310,35	214345	53,55	214345	52,7	214583,0
lin318	5	**212390**	107,96	213400	9,49	213180	765,69	213180	412,97	213180	430,63	212655,0
	7	241957	51,37	243050	17,09	237768	409,76	239442	251,82	239442	254,18	**235254,**
	9	180908	11,15	177524	6,76	**177089**	474,69	177551	53,99	177551	53,23	177474,8
rd400	5	76730	387,91	75990	31,19	76045	1925,31	76085	1260,97	76085	1372,77	**75769,0**
	7	84304	191,73	83149	35,01	**81070**	3341,87	83284	754,66	83284	760,06	82199,0
	9	60854	33,83	59988	23,42	59896	898,73	**59700**	150,97	**59700**	140,96	60163,6
pr439	5	555290	578,09	553280	32,01	563565	2728,81	563820	1937	563895	1931,39	**548853,(**
	7	631219	295,32	628959	87,1	641209	2710,81	627146	1081,01	**623153**	1097,17	639588,4
	9	480415	58,05	473940	32,29	475559	438,34	**473614**	230,32	**473614**	222,43	475886,2
pcb442	5	263185	549,74	**256770**	67,95	258860	2410,99	259065	1778,85	259215	1869,24	260706,0
	7	282737	261,58	277310	36,89	**270561**	1576,52	275336	919,71	275336	927,54	277106,2
	9	204837	87,39	204324	23,75	198332	4171,68	**197521**	374,43	**197521**	380,93	198151,4
d493	5	173385	1066,11	174915	27,92	**173040**	3342,74	173105	2903,59	173185	2961,99	173454,0
	7	193426	439,62	191023	44,21	**189335**	3962,55	189884	1477,74	189884	1558,93	190468,6
	9	157580	98,59	157701	106,28	**152517**	6069,83	156417	389,85	156417	401,73	152644,0

Table 7.10: Heuristic results for MCP1 on instances with more than 200 nodes

Name	β	LAM-E	LAM-t	VNS-E	VNS-t	VNTS-E	VNST-t	TS-E	TS-t	LS-E	LS-t	MS-E	MS-t
eil51	0,08	5,78	0,57	7,23	0,02	8,67	0,25	8,67	0,09	8,67	0,09	**5,61**	0,10
	0,22	**2,85**	0,40	3,66	0,07	6,10	0,38	6,50	0,16	6,50	0,18	4,07	0,17
	0,42	3,51	0,18	3,51	0,04	**1,17**	0,70	7,02	0,11	7,02	0,11	5,26	0,13
st70	0,08	3,43	1,46	2,63	0,16	4,85	1,00	2,63	0,44	2,63	0,46	**2,55**	0,49
	0,22	1,18	0,77	3,54	0,16	**0,59**	2,22	3,24	0,52	3,24	0,54	9,38	0,54
	0,42	12,13	0,34	4,60	0,07	**0,84**	2,76	5,44	0,30	5,44	0,30	4,69	0,34
eil76	0,08	**4,42**	2,07	8,60	0,13	9,09	1,13	6,63	0,46	6,88	0,51	8,26	0,45
	0,22	**2,25**	1,26	14,61	0,19	6,37	1,22	7,49	0,65	5,99	0,71	8,61	0,70
	0,42	8,54	0,47	16,46	0,09	**0,61**	2,67	4,88	0,44	5,49	0,44	7,20	0,41
pr76	0,08	**7,24**	1,99	14,74	0,28	14,75	1,12	13,42	0,68	13,42	0,69	10,25	0,63
	0,22	7,71	1,16	13,42	0,18	**6,87**	1,94	10,24	0,64	10,24	0,68	10,71	0,69
	0,42	9,45	0,48	6,14	0,13	**1,91**	1,74	3,77	0,41	3,25	0,46	7,22	0,35
rat99	0,08	**3,95**	5,81	6,92	0,53	6,81	2,84	6,70	1,30	6,59	1,31	4,87	1,49
	0,22	7,08	2,72	7,08	0,70	**0,67**	11,52	3,20	1,58	3,20	1,68	7,66	1,72
	0,42	17,39	0,86	10,63	0,16	**0,24**	6,57	5,07	0,65	5,31	0,65	5,70	0,64
kroA100	0,08	**1,20**	4,82	11,87	0,98	10,11	3,16	9,99	1,59	9,99	1,76	9,15	1,86
	0,22	8,04	2,03	8,64	0,63	7,84	6,42	**7,39**	1,50	**7,39**	1,60	7,72	1,59
	0,42	14,55	0,90	17,17	0,17	**7,56**	4,46	11,92	0,92	9,87	0,93	11,69	0,74
kroB100	0,08	7,51	4,91	7,93	0,94	8,35	3,56	8,27	1,64	7,59	1,98	**5,82**	1,84
	0,22	6,55	2,27	3,87	0,50	6,45	6,27	**2,56**	1,58	3,07	1,77	7,03	1,74
	0,42	10,72	0,77	10,50	0,25	**5,19**	3,45	11,05	0,62	11,05	0,64	5,22	0,73
kroC100	0,08	**5,26**	4,86	5,32	0,96	5,87	3,21	5,29	1,85	5,56	1,99	6,38	1,87
	0,22	4,25	2,02	9,84	0,58	**1,52**	7,22	8,23	1,45	8,23	1,50	6,07	1,57
	0,42	14,87	0,70	11,34	0,12	**0,11**	6,67	7,14	0,73	7,14	0,75	8,82	0,71
kroD100	0,08	5,75	5,21	5,62	0,65	**3,62**	3,40	3,79	1,57	4,19	1,81	6,03	1,70
	0,22	2,36	2,46	2,93	0,74	**1,27**	8,10	2,79	1,58	2,22	1,63	3,99	1,63
	0,42	12,24	1,00	8,83	0,18	**0,28**	12,11	12,70	0,52	12,70	0,51	5,56	0,70
kroE100	0,08	**4,52**	5,21	9,87	0,65	9,61	2,90	9,61	1,44	9,61	1,58	4,85	1,75
	0,22	6,37	2,35	5,13	0,66	**0,34**	7,66	7,08	1,57	5,51	1,85	4,34	1,70
	0,42	8,47	0,76	4,61	0,22	**0,26**	6,95	4,73	0,80	4,88	0,80	3,88	0,90
eil101	0,08	8,21	5,86	9,94	0,47	10,15	3,06	**7,78**	1,34	9,29	1,34	8,16	1,36
	0,22	8,71	3,35	8,01	0,78	6,97	5,02	6,27	1,72	**5,57**	1,84	10,73	1,78
	0,42	8,38	1,01	4,19	0,27	5,99	8,93	**1,80**	0,94	**1,80**	0,97	14,49	0,94
pr107	0,08	**0,68**	4,88	2,57	0,90	1,02	6,93	2,64	2,18	2,64	2,23	2,14	2,35
	0,22	2,97	0,75	2,33	0,19	**0,28**	6,57	1,41	0,90	1,41	0,93	2,28	0,95
	0,42	1,80	0,15	3,50	0,16	**0,00**	6,81	3,50	0,47	3,50	0,48	2,60	0,47
pr136	0,08	**2,19**	19,24	9,14	2,81	9,31	13,18	8,76	6,53	6,53	7,47	6,26	6,95
	0,22	**3,62**	6,53	9,39	2,10	7,96	51,33	10,29	4,90	9,28	5,48	10,11	4,08
	0,42	12,53	1,66	5,08	0,44	**1,66**	7,56	4,53	1,87	4,53	1,91	4,50	1,54
pr144	0,08	4,93	9,21	1,38	3,26	**0,62**	13,60	0,94	6,43	0,90	7,03	5,66	6,97
	0,22	11,30	5,59	7,01	0,51	**0,19**	31,25	6,51	1,53	6,51	1,56	3,75	2,33
	0,42	0,00	1,49	5,78	0,15	**-0,38**	10,78	1,10	0,93	1,10	0,93	1,92	1,05
kroA150	0,08	**1,56**	21,22	9,40	4,37	5,48	14,19	5,48	8,66	4,93	10,22	5,52	9,60
	0,22	8,53	6,63	7,96	2,37	8,39	27,78	**5,84**	5,54	**5,84**	5,87	9,48	5,21
	0,42	0,00	1,83	1,24	0,49	**-6,91**	11,82	-3,30	1,97	-3,66	2,21	1,14	2,63
kroB150	0,08	7,07	21,57	3,15	3,80	2,77	12,91	**2,65**	7,21	3,89	7,46	6,45	8,87
	0,22	5,78	6,60	5,51	1,91	**0,44**	46,10	3,26	4,20	3,26	4,38	8,63	5,82
	0,42	0,00	1,29	5,23	0,52	**-1,60**	23,15	3,93	1,52	3,93	1,53	2,29	1,98
pr152	0,08	0,00	9,98	-0,78	3,39	**-2,44**	26,80	-0,96	6,38	-1,68	7,25	-1,28	7,39
	0,22	0,00	2,05	0,36	0,40	**-5,58**	22,43	-3,48	3,01	-3,48	3,20	-2,84	2,27
	0,42	0,00	0,32	-4,27	0,19	**-6,73**	9,64	-5,50	1,19	-5,50	1,21	-4,68	1,18
rat195	0,08	0,00	70,40	-0,60	11,08	**-0,84**	41,47	-0,06	17,19	-0,72	19,51	-0,47	20,05
	0,22	5,29	20,74	14,50	5,41	**2,76**	68,66	11,97	11,87	14,27	10,94	9,78	12,27
	0,42	0,00	2,42	-1,83	1,61	**-5,65**	26,11	-3,49	3,92	-3,49	3,89	-0,73	4,21
kroA200	0,08	9,08	60,11	10,82	16,02	**7,61**	62,56	12,13	23,82	11,56	27,29	9,40	27,60
	0,22	0,00	16,58	-5,35	6,61	**-8,91**	49,40	-1,53	14,23	-2,58	13,33	-6,49	14,34
	0,42	0,00	2,04	-1,56	1,86	**-5,55**	41,43	-3,13	5,54	-3,13	5,55	0,30	4,00
kroB200	0,08	8,13	61,43	6,00	16,12	**5,98**	52,97	6,51	23,41	6,36	27,27	7,22	28,33
	0,22	0,00	17,34	3,55	5,39	**-1,53**	51,04	0,71	14,30	1,68	12,69	1,75	14,98
	0,42	0,00	2,51	4,57	1,58	**-5,58**	44,49	0,40	4,92	0,40	4,93	0,90	5,23

Table 7.11: Heuristic results for MCP2 on instances with at most 200 nodes

Name	β	LAM	LAM-t	VNS	VNS-t	VNTS	VNTS-t	TS	TS-t	LS	LS-t	MS	M
ts225	0,08	**84824**	100,33	88540	27,82	86552	78,88	91241	31,94	91241	35,07	90764,4	37
	0,22	64003	26,01	61529	7,4	**56409**	188,8	60906	15,24	61022	15,48	61303,6	14
	0,42	39334	3,64	41169	0,99	**36798**	101,71	38622	4,58	38622	4,64	39647,6	4,
tsp225	0,08	**2458**	100,64	2576	21,34	2535	57,1	2551	35,5	2552	38,52	2541,6	39
	0,22	1473	18,44	1424	8,12	1443	81,71	**1393**	18,01	1398	18,34	1438,2	16
	0,42	1011	1,67	1004	1,05	**929**	57,93	963	4,27	963	4,28	998	3,
pr226	0,08	50195	31,39	49498	6,53	**47577**	141,73	49109	17,78	49223	17,18	49488,2	20
	0,22	43351	2,15	39209	1,6	**38147**	46,1	38911	5,91	38911	5,61	38935,4	6,
	0,42	40192	0,69	34973	0,44	**34067**	12,99	34953	2,76	34953	2,71	34621,2	2,
pr264	0,08	**31207**	141,71	31835	39,82	31920	88,16	31711	63,49	31709	68,51	32250,4	69
	0,22	21444	10,65	22114	5,76	**21393**	188,69	21763	18,36	21945	17,35	21815,8	15
	0,42	20224	1,05	19168	1,02	**18383**	50,71	18529	8,58	18529	8,71	18825,6	6,
a280	0,08	**1590**	250,89	1695	61,64	1678	163,35	1703	84,06	1698	92,12	1656	87
	0,22	893	33,15	934	18,15	**846**	242,79	906	34,92	928	35,42	918,4	33
	0,42	640	2,86	622	2,29	**579**	93,44	592	9,13	594	8,77	602,6	8,
pr299	0,08	**29021**	298,07	30856	74,73	30471	171,12	30559	104,98	30501	121,57	30058,6	115
	0,22	19517	61,01	17977	26,2	**17354**	374,46	17780	42,15	18016	41,19	18181,2	49
	0,42	14308	4,61	13252	4,62	**12825**	74,93	13316	13,49	13316	13,57	13198,6	12
lin318	0,08	25394	332,48	25238	81,64	**24152**	281,62	24835	123,22	24935	123,68	25222,8	128
	0,22	14261	63,83	14215	13,65	**13581**	148,72	13954	29,75	13854	34,87	14223,4	35
	0,42	10058	2,62	9181	2,88	**8823**	106,6	9033	9,09	9020	9,17	9084	11
linhp318	0,08	25394	332,77	25238	81,36	**24152**	281,73	24835	122,48	24935	123,67	25222,8	129
	0,22	14261	63,6	14215	13,62	**13581**	148,84	13954	29,69	13854	34,87	14223,4	35
	0,42	10058	2,62	9181	2,89	**8823**	106,62	9033	9,11	9020	9,16	9084	11
rd400	0,08	8883	1211,1	8864	339,53	**8568**	1521,63	8678	521,82	8853	493,15	8803,4	507
	0,22	5174	220,55	4742	76,31	**4545**	529,53	4587	171,19	4589	166,71	4741	124
	0,42	2762	12,03	2693	8,86	**2497**	325,56	2638	25,58	2638	25,01	2629,6	30
pr439	0,08	**64745**	1625,49	70478	603,61	67682	3049,14	68775	883,55	70694	777,97	68207,6	803
	0,22	46043	554,7	41122	62,27	**37089**	1552,06	39518	152,24	39364	158,85	39336,2	255
	0,42	23891	16,89	22493	27,94	**21556**	1129,57	22074	63,94	22170	62,7	23045,6	55
pcb442	0,08	28015	1853,95	29056	495,65	**27830**	3072,24	29077	669,54	28992	734,29	28239	713
	0,22	13712	210,29	14181	67,17	**13588**	940,41	13801	138,93	13764	137,55	14063,8	178
	0,42	10150	12,91	9418	12,54	**8823**	271,41	9355	31,94	9229	32,83	9452	35
d493	0,08	20900	2999,7	20779	957,26	**20715**	4490,02	21176	1138,89	21160	1209	20986,8	123
	0,22	14155	716,08	13131	100,37	**12278**	2609,97	13069	246	13069	241,31	12846	319
	0,42	10048	25,31	9529	20,5	**8837**	2987,34	9416	79,26	9414	79,06	9148,2	62

Table 7.12: Heuristic results for MCP2 on instances with more than 200 nodes

Cycle size	MCP1-200	MCP1-500	MCP2-200	MCP2-500
big	45 %	25 %	30 %	50 %
medium	55 %	33,3 %	65 %	91,6 %
small	65 %	66,6 %	95 %	100 %
Average:	55 %	41,6 %	63,3 %	80,5 %

Table 7.13: Summary of results for VNTS

Appendix A

ABACUS implementation

The branch-and-cut algorithm described in Chapter 5 was implemented in C++, using ABACUS 2.2 and the LP-solver CPLEX 6.0. ABACUS is a collection of C++ libraries for the implementation of branch-and-cut and branch-and-price and the combination of both techniques. It applies the concept of object oriented programming. As a result, an implementation of a problem specific algorithm is obtained by deriving some classes from abstract base classes of ABACUS in order to embed problem specific functions.

In this chapter we describe the main features of our ABACUS based branch-and-cut algorithm for the MCP.

The classes

In order to use ABACUS for a new application we have to derive problem specific classes from some base classes. Usually only four base classes of ABACUS are involved: ABA_VARIABLE, ABA_CONSTRAINT, ABA_MASTER and ABA_SUB. A problem specific algorithm can be composed by simply defining some pure virtual functions of the base classes in the derived classes and redefining some virtual functions.

Figure A.1 shows the inheritance tree of the classes of our branch-and-cut algorithm for MCP. The problem specific classes are surrounded by a bold frame.

In the following sections we will describe briefly some details of the implementation of these classes.

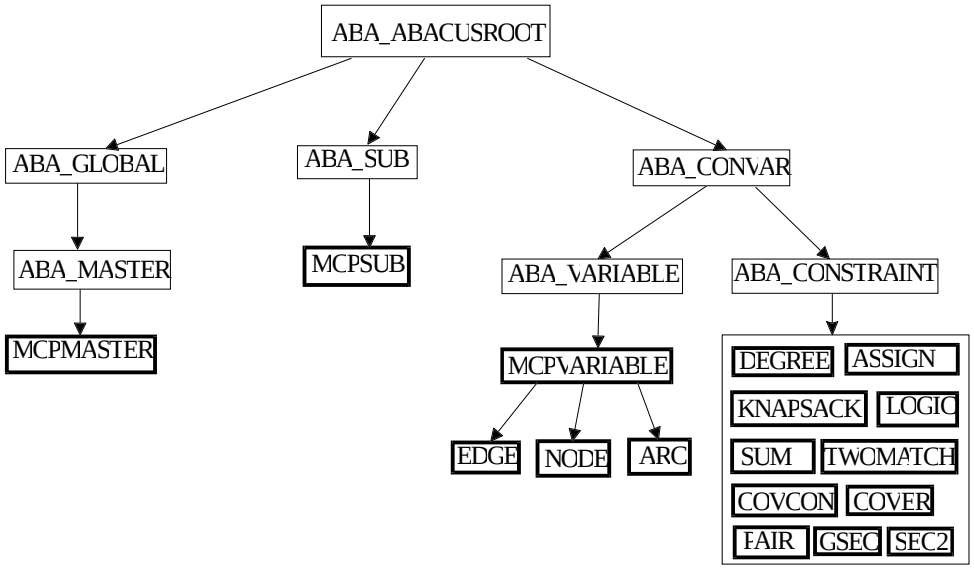

Figure A.1: The MCP inheritance tree

The variables

To simplify the implementation we have used three classes to represent the variables within the branch-and cut-algorithm. These are the NODE, EDGE and ARC classes, that correspond to the y_{ii}, x_{ij} and y_{ij} variables respectively. We derive the three classes from the generic class MCPVARIABLE, that derives from the abstract class ABA_VARIABLE. This intermediate class is used in order to determine correctly the coefficients of the different types of variables within the different types of constraints.

The function in charge of giving the coefficient of a variable within a constraint is *coeff()*. This is a pure virtual function of the class ABA_CONSTRAINT, and so it has to be defined in each constraint class MYCONSTRAINT. In our case, we define it in the following way:

```
double MYCONSTRAINT::coeff(ABA_VARIABLE*v)
{
  MCPVARIABLE*var=(MCPVARIABLE*)v;
  return (var->myconstraint_coeff(...));
}
```

The function *myconstraint_coeff()* is declared as virtual in the class MCPVARIABLE, and it is redefined in the appropriate way inside each of the classes NODE, EDGE and ARC.

All three type of variables are defined as binary.

The constraints

We have derived from the abstract class ABA_CONSTRAINT a class for each of the constraints that define our problem.

ABACUS allows to distinguish between *globally* and *locally valid* constraints.. In our algorithm all the constraints are defined as globally valid. This means that they are valid not only in one specific subproblem but in all the branch-and-cut tree. Moreover, the constraints can be defined as *static* or *dynamic*. Static constraints stay always in the LP once they are introduced, while dynamic constraints can be removed and inserted again when needed.

When implementing a constraint class it is important to think in the trade-off between performance and memory usage. A memory efficient storage format can be one of the keys for the solution of large instances. However, such formats are in general not very useful when computing the variable coefficients within the constraint. On the other hand, a different format, that uses more memory, might make much quicker the computation of the variable coefficients. To cope with this fact, ABACUS gives the possibility of defining two formats for the constraints: *compressed* and *expanded*. The expanded format is used when the coefficient computation is performed. Then, the constraint is compressed. The implementation of the expansion and the compression is optional.

We describe now briefly the constraint classes used in our branch-and-cut algorithm.

DEGREE

The class DEGREE corresponds to the degree constraints (3.2). These constraints are declared static and are represented internally as $x(\delta(v_i)) - 2y_i = 0$. A variable associated with an edge e has value 1 if and only if v_i is one of the endnodes of e. The variable associated with node v_i has value -2.

ASSIGN

The class ASSIGN corresponds to the assignment constraints (3.4). These constraints are declared static. They are defined by the corresponding node v_i.

KNAPSACK

The class KNAPSACK correspond to the constraint (3.20). This constraint is declared static

LOGIC

The class LOGIC represents the logical constraints (3.17). These constraints are declared static.

SUM

The class SUM correspond to the constraint (5.1). This constraints is declared static.

GSEC

The GSEC class correspond to the generalized subtour eliminations constraints (3.16). A GSEC constraints is defined by a cut $S \subseteq V \backslash \{v_1\}$ and a vertex $v_i \in S$. They are represented internally as $x(\delta(S)) - 2 \sum_{v_j \in S} y_{ij} \geq 0$. In the compressed format we store the vertices of S in an array. An edge variable x_{hk} has coefficient 1 if and only if v_h and v_k are both in the array S. An arc variable y_{hk} has coefficient -2 if and only if $h = i$ and v_k belong to S. Therefore, the computation of an edge or arc variable coefficient in the compressed format requires $O(|S|)$ time. We also provide an expanded format, represented by a boolean array of size $|V|$. The component j of the array is *true* if and only if the vertex v_j is contained in the cut S. With this format, the computation the coefficient of an edge or arc variable only requires $O(1)$ time.

The GSEC constraints are declared dynamic.

PAIR

This class corresponds to constraints (3.15). These constraints are declared dynamic. They are defined by a pair of vertices v_i and v_j.

COVCON

The class COVCON corresponds to the cover-connectivity constraints (3.26), which are declared dynamic and are internally represented as $x(\delta(S)) - 2\sum_{(v_i,v_j)\in\bar{T}} y_{ij} \geq 2(1-|T|)$. These constraints are defined by the cut S, the number $|T|$ of original arcs, and the set $\bar{T}$ of arcs. The set $\bar{T}$ is represented by arrays of size $|\bar{T}|$, one containing the tails ($Ttail$) and other the heads ($Thead$) of the arcs. An arc variable will have coefficient -2 if and only if the its tail and its head are in the same position in the arrays $Ttail$ and $Thead$, respectively. In the expanded format, the cut S is represented by a boolean array, as for the GSEC. So, the computation of an edge variable coefficient takes $O(|S|)$ time in the compressed format and constant time in the expanded format. The computation of an arc variable coefficient always requires $O(|\bar{T}|)$ time.

COVER

This class corresponds to the cover constraints (3.21). These inequalities are declared dynamic. They are defined by the number $|T|$ of original arcs, and the set $\bar{T}$ of arcs. The set $\bar{T}$ is represented as in the cover-connectivity constraints. No expanded format is given.

SEC2

The class SEC2 corresponds to the constraints (3.23). These constraints are defined dynamic. Each of them is defined by a set of nodes S. This set is represented in the compressed and expanded format as the cut in the GSEC.

TWOMATCH

The class TWOMATCH correspond to the 2-matching constraints (3.18). These inequalities are declared dynamic, and they internally are represented as $x(E(H)) + x(T) - y(H) \leq (|T|-1)/2$. In the compressed format we represent the handle by an array H of vertices. The teeth are represented by to arrays, $Ttail$ and $Thead$, that contain in the i-th position the tail and the head of the i-th tooth, respectively. With this format, to calculate the coefficient of a variable requires $O(|H|)$ time if it is a node variable, $O(|H|+|T|)$ time if it is an edge variable, and $O(|T|)$ time if it is an arc variable. In the expanded format, the handle is represented as the cut in

the GSEC constraints (that is, using an array of boolean), and the teeth are represented by two array of integer, $eTtail$ and $eThead$, of size $|V|$. The teeth are numerated from 1 to $|T|$. These two arrays are initialized to -1. Then, the positions t and h of the arrays $eTtail$ and $eThead$ receive the number j if (v_t, v_h) is the j-th teeth. With the expanded format, the computation of all the coefficients requires only a constant time.

The master

The class MCPMASTER is derived from the abstract base class ABA_MASTER. The main two reasons to require a problem specific master of the optimization are that we have to embed the problem specific data, and that we have to initialize the first subproblem, i.e., the root node of the branch-and-cut tree.

The constructor

The constructor of the class MCPMASTER calls the constructor of its base class ABA_MASTER. The arguments are the problem name and the value α, used to calculate the routing and assignment costs in MCP1 and the upper bound d_0 in MCP2.

```
MCPMASTER::MCPMASTER(const char*problemName, char *alfa):
  ABA_MASTER(problemName,true,false,ABA_OPTSENSE::Min,0.001,0.00001)
```

Regarding the arguments of the ABA_MASTER constructor, the first one is again the problem name. The second argument is *true* because we use cutting plane generation for the solution of the subproblems. Since no variables are generated dynamically the third argument is set to *false*. The forth argument indicates the sense of the optimization (the MCP is a minimization problem). The last two arguments are the *eps* and *machineEps*, respectively. The reason for having two different zero-tolerance is that their commitment is different: *eps* is used to test if a constraint is violated, while *machineEps* is used to compare the value of a floating point variable with 0. The default values of this two parameters are $eps = 1.0 \times 10^{-4}$ and $machineEps = 1.0 \times 10^{-7}$.

In the body of the constructor the problem data is read and the memory is allocated.

First subproblem

The root of the branch-and-cut tree is initialized with an object of the class MCPSUB, that is derived from the class ABA_SUB. This initialization must be performed by a definition of the function *firstSub()*, which returns a pointer to the first subproblem.

```
ABA_SUB *MCPMASTER::firstSub()
{
  return new MCPSUB(this);
}
```

Initialization of the constraints and variables

The constraints and variables that are not dynamically generated must be set up and inserted in pools in a member function of the class MCPMASTER. For this purpose we use the virtual dummy function *initializeOptimization()*, which is called at the beginning of the optimization.

By default ABACUS provides three different pools: one for the variables and two for the constraints. The first constraint pool stores the constraints that are not dynamically generated and with which the first LP-relaxation of the first subproblem is initialized. The second constraint pool is empty at the beginning and it is filled up with dynamically generated cutting planes. We have called the variable pool *variables*, and the first constraint pool *initConstraints*. In *initConstraints* we store the constraints of classes DEGREE, ASSIGN, PAIR, KNAPSACK and SUM, which gives *3n+1* constraints for an instance with n nodes. The size of the variable pool is set to *nVariables = n(3n-1)/2*, which is the total number of x and y variables for an instance with n nodes. Finally, we set the size of the cut pool to 10000. The pools are initialized by calling the function *initializePools()*. In our case, the call is:

```
initializePools(initConstraints,variables,nVariables,10000);
```

After the pools are set up, the primal bound is initialized using function

primalBound() with the value of a feasible solution returned by the function *mcpHeuristic()*.

```
double length= mcpHeurist(succ);
primalBound(length);
```

The parameters

The function *initializeParameters()* redefines a virtual dummy function of the class ABA_MASTER. This function is called at the beginning of the optimization.

```
void MCPMASTER::initializeParameters()
{
  readParameters(".mcp");
  int status= getParameter("ShowBestTour",showBestTour_);
  if(status)
  {
    err()<<"Parameter ShowBestTour not in configuration file .tsp."<<endl;
    exit(Fatal);
  }
}
```

The function *readParameters()* reads the file *.mcp* that contains the parameters and theirs values, in the ABACUS parameter file format. In our case, in the file *.mcp* we first redefine the following parameters form the file *.abacus*.

ObjInteger	true
MaxConAdd	300
MaxConBuffered	300
TailOffNLps	5
TailOffPercent	0.0025
EliminateFixedSet	true
OutputLevel	Statistics
MaxCpuTime	00001:59:59
NBranchingVariableCandidates	5
CplexPrimalPricing	CPX_PPRIIND_STEEPQSTART

Then we introduce an extra parameter.. At the end of the optimization the

best known tour is output if the value of the parameter *ShowBestTour* is true.

ShowBestTour true

The subproblem

The class MCPSUB is derived from the abstract class ABA_SUB. It implements the specific functions for the optimization of a subproblem (node of the branch-and-cut tree).

Besides the constructor, only other two virtual functions of the class ABA_SUB have to be defined, those that check the feasibility of the solution given by the LP-relaxation, and generate the new nodes after the branching step. Moreover, this class provides the functions for performing the cutting plane generation and improving the upper bound by using primal heuristics.

The constructors

The class ABA_SUB has two different constructors: one for the root node and one for all other nodes of the optimization. Therefore, we also have to implement these two constructors for the class MCPSUB.

The root node constructor for the class ABA_SUB must be called from the root node constructor of the class MCPSUB:

```
MCPSUB::MCPSUB(ABA_MASTER *master):
ABA_SUB(master,10.0,0.0,10.0)
{}
```

The parameters indicate that 10% of additional storage space for constraints, no additional space for variables and 10% additional space for nonzeros of the constraint matrix in the LP-solver, should be allocated.

The constructor for the son of an existing node takes all the default values.

The feasibility check

After the LP-relaxation is solved we have to check if its optimum solution is a feasible solution for our optimization problem. Therefore, we have to define the pure virtual function *feasible()* in that class MCPSUB, which returns *true* is the LP-solution is a feasible solution and *false* otherwise.

The LP-solution is MCP feasible if all variables have value zero or one, $y_{ij} \leq y_{jj}$ for all pair of nodes $\{v_i, v_j\}$, and the graph induced by the edge variables x_{ij} with value one is connected (i.e., there are not subtours).

We check if the graph induced by the edge variables with value one is connected with the help of a disjoint set data structure (class ABA_FASTSET). Initially each node is a set by itself. Then all edges are scanned. If an edge variable has value one we check is its two endnodes are contained in disjoint sets. If this is the case, we merge the two sets and continue. Otherwise a subtour have been found, providing that the scanned edge is not the last one that close the tour,

The separation

The MCP violated cuts are generated by redefining the virtual dummy function *separate()*. ABACUS allows to distinguish between the separation from scratch (that performed examining the LP-solution to see if it violates a given type of cut) and the separation from a constraint pool. Newly generated constraints have to be added by the function *addCons()* to a buffer of the class ABA_SUB, before passing to the cut pool. Constraints generated in earlier iterations that have become inactive in the meantime might be still contained in the cut pool. These constraints can be regenerated by calling the function *constraintPoolSeparation()*, which adds the constraints to the buffer without the explicit call of the function *addCons()*.

An a priori good separation strategy consists of generating new cuts from scratch only when pool separation fails. We tried first this approach for the MCP, but the computational results showed that, for this problem, the pool separation increases the running time. Therefore, we opted for performing only separation from scratch.

The separation procedures for the different kind of cuts have been de-

scribed in Section 5.1. We applied them in the order mentioned in Section 5.2. We go on with the separation of a type of constraints only if the number of violated cuts generated until that moment does not exceed the value given by the parameter *MaxConAdd*, that we set to 300.

Primal heuristic

After the LP-relaxation has been solved in the subproblem optimization, the virtual function *improve()* is called. This function allows to try to improve the current primal bound by using the heuristic procedures described in Section 5.2.6. If the value returned by the heuristic is better than the current upper bound, then this is updated.

Branching strategy

ABACUS gives the possibility of choosing the best variable for branching among a number of candidates (this is called *strong branching*). The number of variables to test is fixed with the parameter *NBranchingVariableCandidates*, that we set to 5. The five candidate variables are chosen by redefining the virtual function *selectBranchingVariableCandidates()*.

For the MCP, we take as candidates for branching the five node variables with fractional values closest to 0.5. If there are not five fractional node variables, we select with the same criterium the necessary edge variables.

Associated to each candidate variable there are two branching rules: one consisting of fixing it to zero, and other consisting of fixing it to one. To determine the best branching variable, each rule of each candidate variable is given a rank. In the default implementation (the one we have used), this rank is obtained by performing a limited number of iterations of the dual simplex method for the first linear program of the subproblem defined by the rule. Then, the candidate variable for which the minimal rank of its rules is maximal is selected for branching.

The main program

The main program creates an object of the class MCPMASTER and begins the optimization. The two arguments required by the constructor of the MCPMASTER class are given in the command line.

The program returns 0 if the optimization finishes correctly, and 1 otherwise.

```
#include <stdio.h>
#include <iostream.h>
#include <stdlib.h>
#include "mcpmaster.h"
int main(int argc,char**argv)
{
  if(argc!=3)
  {
    cerr<<"usage: "<<argv[0]<<" <file>"<<endl;
    return 1;
  }
  MCPMASTER *mcp= new MCPMASTER(argv[1], argv[2]);
  ABA_MASTER::STATUS status= mcp->optimize();
  delete mcp;
  if(status)
    return 1;
  else
    return 0;
}
```

Bibliography

[1] P. Avella, A. Sforza. The Median Path Problem: formulations and polyhedral analysis. Working paper, 1999.

[2] P. Avella, A. Sforza. The Median Path Problem: a computational study. Working paper, 1999.

[3] E. Balas. The Prize Collecting Traveling Salesman problem, *Networks* 19 (1989), 621–636.

[4] E. Balas, A. Ho. Set covering algorithms using cutting planes, heuristics and subgradient optimization: a computational study, *Mathematical Programming* 12 (1980), 37–60.

[5] P. Bauer. The Circuit Polytope: Facets, *Mathematics of Operations Research* 22 (1997), 110–145.

[6] C. Berge. *Graphs and hypergraphs.* North-Holland, Amsterdam, 1973.

[7] J.A. Bondy, U.S.R. Murty. Graph theory with applications. University Press, Belfast, 1976.

[8] N. Christofides. *Graph Theory. an algorithmic approach.* Academic Press, New York, 1975.

[9] Th. Christof. *Ein Verfahren zur transformation zwischen polyederdarstellungen.* Diplomarbeit. Universität Augsburg, 1991.

[10] R. Church, J. Current. Maximal covering tree problems, *Naval Research Logistics* 40 (1993) 129–142.

[11] J. Current. The design of a hierarchical transportation network with transshipment facilities, *Transportation Science* 22 (1988), 270-277.

[12] J. Current, J.L. Cohon, C.S Revelle. The shortest covering path problem: an application of locational constraints to network design, *Journal of Regional Science* 24 (1984), 161-183.

[13] J. Current, H. Pirkul, E. Rolland. Efficient algorithms for solving the shortest covering path problem, *Transportation Science* 28 (1994), 317-327.

[14] J. Current, C.S. Revelle, J.L. Cohon. The maximum covering/shortest path problem: a multiobjective network design and routing formulation, *European Journal of Operational Research* 21 (1985) , 189-199.

[15] J. Current, C.S. Revelle, J.L. Cohon. The median shortest path problem: a multiobjective approach to analyze cost vs. accessibility in the design of transportation networks, *Transportation Science* 21 (1987), 188-197.

[16] J. Current, D.A. Schilling. The covering salesman problem, *Transportation Science* 23 (1989), 208-213.

[17] J. Current, D.A. Schilling. The median tour and maximal covering tour problems: formulation and heuristics, *European Journal of Operational Research* 73 (1994), 114-126.

[18] M. Ehrgott, J. Fretag, H.W. Hamacher, F. Maffioli. Heuristics for the K-cardinality tree subgraph problems. Working paper, 1995.

[19] M. Fischetti, J.J. Salazar, P. Toth. The symmetric generalized travelling salesman polytope, *Networks* 26 (1995), 113-123.

[20] M. Fischetti, J.J. Salazar, P. Toth. A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem, *Operations Research* 45 (1997), 378–394.

[21] M. Fischetti, J.J. Salazar, P. Toth. Solving the Orienteering Problem through Branch-and-Cut, *INFORMS Journal on Computing* 10 (1998), 133–148.

[22] R.W. Floyd. Algorithm 97: shortest path, *Communications of ACM* 5 (1962), 345.

[23] M.R. Garey, D.S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[24] M. Gendreau, A. Hertz, G. Laporte. New insertion and postoptimization procedures for the Traveling Salesman Problem, *Operations Research* 40 (1992), 1086-1094.

[25] M. Gendreau, G. Laporte, F. Semet. The Covering Tour Problem, *Operations Research* 45 (1997), 568–576.

[26] M. Gendreau, G. Laporte, F. Semet. The Selective Traveling Salesman Problem, *Networks* 32 (1998), 263–273.

[27] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing* 1 (1989), 190-206.

[28] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing* 2 (1990), 4-32.

[29] A.J. Goldman. Optimal center location in simple networks, *Transportation Science* 5 (1971), 406-409.

[30] M. Grötschel, L. Lovász, A. Schrijver. *Geometric algorithms and combinatorial optimization*, Springer-Verlang, 1988.

[31] M. Grötschel, M.W. Padberg. Polyhedral Theory, in *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds), Wiley, Chichester, 251–305, 1995.

[32] Z. Gu, G.L. Nemhauser, M.W.P. Savelsbergh. Cover Inequalities for 0-1 Integer Programs: Computation, *INFORMS Journal on Computing* 10 (1998), 427–437.

[33] S.L. Hakimi, E.F. Schmeichel, M. Labbé. On locating path- or tree-shaped facilities on networks, *Networks* 23 (1993), 543-555.

[34] P. Hansen, N. Mladenovic, D. Pérez-Brito. Variable Neighborhood Decomposition Search. To appear in *Jornal of Heuristics* (1999).

[35] J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, Michigan (1975).

[36] V.A. Hutson, C.S. Revelle. Maximal direct covering tree problems, *Trasportation Science* 23 (1989) , 188-299.

[37] V.A. Hutson, C. Revelle. Indirect covering tree problems on spanning tree networks, *European Journal of Operational Research* 65 (1993), 20-32.

[38] D. Johnson, C. Papadimitriou. Computational Complexity. In *The Traveling Salesman Problem . A guided tour of Combinatorial Optimization*, E. Lawler, J. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds), Wiley, Chichester, 1985.

[39] M. Jüenger, S. Thienel. Introduction to ABACUS - A Branch-And-CUt System, Technical Report No. 97.263, Universiät zu Köln, 1997.

[40] T.U. Kim, T.J. Lowe, A. Tamir, J.E. Ward. On the Location of a Tree-shaped Facility, *Networks* 28 (1996), 167–175.

[41] T.U. Kim, T.J. Lowe, J.E. Ward. Locating a median subtree on a network, *INFOR* 29 (1991), 153-166.

[42] T.U. Kim, T.J. Lowe, J.E. Ward, R.L. Francis. A minimum length covering subgraph of a network, *Annals of Operations Research* 18 (1989), 245-260.

[43] T.U. Kim, T.J. Lowe, J.E. Ward, R.L. Francis. A minimum-length covering subtree of a tree, *Naval Research Logistics* 37 (1990), 309-326.

[44] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi. Optimization by simulated annealing, *Science* 220(1983), 671-666.

[45] M. Labbé, G. Laporte. Maximizing user Convenience and Postal Service Efficiency in Post Box Location, *Belgian Journal of Operations Research, Statistics and Computer Science* 26 (1986), 21–35.

[46] M. Labbé, G. Laporte, I. Rodríguez Martín. Path, Tree and Cycle Location. In *Fleet Management and Logistics*, T.G. Crainic and G. Laporte (eds), Kluwer, Boston, 1998.

[47] M. Labbé, G. Laporte, I. Rodríguez Martín, J.J. Salazar. The Median Cycle Problem. C.R.T. working paper, 1999.

[48] S. Lin, B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research* 21 (1973), 498-516.

[49] F. Maffioli. Finding the best subtree of a tree, technical report 91.041 (1991) , Politecnico di Milano.

[50] J.A. Mesa, T.B. Boffey. A review of extensive facility location in networks, *European Journal of Operational Research* 0 (1996), 1-12.

[51] E. Minieka. The Optimal Location of a Path or Tree in a Tree Network, *Networks* 15 (1985), 309–321.

[52] E. Minieka, N.H. Patel. On finding the core of a tree with a specified length, *Journal of Algorithms* 4 (1983), 345-352.

[53] N. Mladenovic. A variable neighborhood algorithm - a new metaheuristic for combinatorial optimization. Presented at Optimization Days, Montreal (1995).

[54] N. Mladenovic, P. Hansen. Variable Neighborhood Search, *Computers Oper. Res.* 24 (1997), 1097–1100.

[55] N. Mladenovic, J.A. Moreno-Pérez, J.M. Moreno-Vega. A Chain-Interchange Heuristic Method, *Yugoslav J. Oper. Res.* 6 (1996) 41–54.

[56] G.L. Nemhauser, P. Vance. Lifted Cover Facets of the 0-1 Knapsack Polytope with GUB Constraints *Operations Research Letters* 16 (1994), 255–263.

[57] G.L. Nemhauser, L.A. Wolsey. *Integer and combinatorial optimization.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1988.

[58] M.W. Padberg, M.R. Rao. Odd minimum cut-sets and b-matchings, *Mathematics of Operations Research* 7 (1982), 67-80.

[59] M.W. Padberg, G. Rinaldi. Optimization of a 532-city Symmetric Taveling Salesman Problem, *Operations Research Letters* 6 (1987), 1-7.

[60] C. Papadimitriou, K. Steiglitz. Combinatorial Optimization: algorithms and complexity. Prentice-Hall, New Jersey, 1982.

[61] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library, *ORSA Journal on Computing* 3 (1991), 376–384.

[62] M.B. Richey. Optimal Location of a Path or Tree on a Network with Cycles, *Networks* 20 (1990), 391–407.

[63] S. Sahni. General techniques for combinatorial approximations, *Operations Research* 25 (1977), 920-936.

[64] A. Schrijver. Theory of Linear and Integer Programming. John Wiley & Sons, Chinchester, 1986.

[65] R.E. Steuer. *Multiple criteria optimization: theory, computation, and application*, Wiley series in probability and mathematical statistics-applied, Wiley, 1986.

[66] A. Tamir. On the complexity of some classes of location problems, *Transportation Science* 26 (1992), 352-354.

[67] A. Tamir. Fully polynomial approximation schemes for locating a tree-shaped facility: a generalization of the knapsack problem. Working paper, Tel-Aviv University.

[68] C. Toregas, C. Revelle. Optimal location under time or distance constraints, *Papers of the Regional Science Association* 28 (1972), 133-143.

[69] C. Toregas, C. Revelle. Binary logic solutions to a class of location problems, *Geographical Analysis* 5 (1973), 145-155.

[70] L.A. Wolsey. Valid Inequalities for 0-1 Knapsacks and MIPS with Generalized Upper Bound Constraints, *Discrete Applied Mathematics* 29 (1990), 251-261.

[71] L.A. Wolsey. Integer programming. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1998.

[72] J. Xu, S.Y. Chiu, F. Glover. Optimizing a ring-based private line telecommunication network using Tabu Search. *Management Science* 3 (1999), 330-345.