Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

Corso di Laurea Magistrale in Ingegneria Matematica

Bottleneck Optimum Communication Spanning Tree Problems

Relatori:
Prof. Elena Fernández
Prof. Edoardo Amaldi

Tesi di laurea di:
Valentina Quintarelli
Matr. 804162

Anno Accademico 2014/2015

Abstract

In recent years, much interest has been focused on various problems that arise in the area of network design. We consider the Optimum Communication Spanning Tree Problem (OCSTP), which is defined as follows. Given a complete undirected graph with a cost associated to each edge and a communication requirement between each pair of nodes, find a spanning tree connecting all the nodes which minimizes the total communication cost, i.e. the sum of the communication costs over all pairs of nodes. This classic problem has been extensively studied in the literature and it is not applied only in the network design area for transportation planning and communication system planning, but also, for example, in the alignment of genomic sequences and in hub allocation.

In this work we investigate two variants of the OCSTP, with a different objective function. In the Minimum Path Optimum Communication Spanning Tree Problem (MP-OCSTP) the objective is to find a spanning tree where the cost of most expensive path between a pair of nodes is minimum. In the Minimum Edge Optimum Communication Spanning Tree Problem (ME-OCSTP) the objective is to find a spanning tree where the cost of the most expensive edge is minimum. These problems are relevant in some applications.

First, we study some special cases of MP-OCSTP in which the optimal solutions have a known structure. In particular, we show that, if the costs and the requirements are all equal, any optimal solution is a star-tree.

Then, we propose three Mixed Integer Linear Programming models for these problems. The main feature that distinguishes the formulations is the number of indeces of the considered variables: 4-index, 3-index and 2-index. The computational results obtained on benchmark instances from the literature show that a large computing time is required to solve the ME-OCSTP to optimality. Instead, to solve the MP-OCSTP the computational times are quite reasonable. After evaluating the quality of the linear relaxation bounds, we also propose

some valid inequalities for the different formulations. When necessary, the valid inequalities are generated by solving the corresponding separation problem. The computational results indicate that for MP-OCSTP all the formulations provide good solutions with an equivalent objective function value for all the instances satisfying the triangular inequality, while the 4-index formulation outperforms the other two formulation on randomized instances. For the ME-OCSTP the linear relaxation of 2-index formulation is more competitive than the other two, but this problem turn out to be very challenging. In most of the cases the valid inequalities do not lead to substantial improvements.

Finally, we present some heuristic algorithms. Greedy algorithms are based on the minimum spanning tree, or on the star-trees or on the Gomory-Hu tree. The local-search algorithm for the MP-OCSTP tries to connect directly the end-nodes of the most expensive path and to delete one of the edges in the generated cycle. Instead, for the ME-OCSTP the local-search algorithm tries to delete the most expensive edge and to add one of the edges that connects the two disconnected components. The results show that the heuristic combining the greedy algorithm based on the Gomory-Hu tree and the local-search algorithm, provides the best upper bound on many instances for both problems. We also develop a randomized version of the algorithm based on the Gomory-Hu tree, which yields improved solutions.

Sommario

Negli ultimi anni, un crescente interesse è stato rivolto a problemi appartenenti all'area del network design. Considereremo l'Optimum Communication Spanning Tree Problem (OCSTP), che è definito come segue. Dato un grafo completo non direzionato con una funzione Costo definita per ogni arco e una funzione Domanda definita per ogni coppia di nodi, si vuole costruire lo spanning tree che minimizza il costo totale di comunicazione, ottenuto come la somma di tutti i costi dell'albero. Questo problema è stato ampiamente studiato e trova applicazione non solo nell'area del network design per la pianificazione di trasporti o di sistemi di comunicazione, ma anche, per esempio, nell'allineamento di sequenze di DNA e nell'allocazione di Hub.

In questa tesi presentiamo due possibili varianti dell'OCSTP. Nel Minimum Path Optimum Communication Spanning Tree Problem (MP-OCSTP) si vuole minimizzare il costo del cammino più costoso per connettere una coppia di nodi. Invece, l'obiettivo del Minimum Edge Optimum Communication Spanning Tree Problem (ME-OCSTP) è quello di minimizzare il costo dell'arco più costoso. Queste due varianti hanno rilevanza in alcune applicazioni.

Innanzitutto, proporremo uno studio della struttura che la soluzione assumerebbe sotto alcune ipotesi particoli. Per quanto riguarda il MP-OCSTP mostreremo che quando i costi e le domande sono tutti uguali tra loro ogni soluzione ottima assume la forma di una stella.

Poi, presenteremo tre modelli di programmazione mista intera. Ogni modello si distingue dagli altri per il numero di indici delle variabili scelte: 4-indici, 3-indici e 2-indici. Per quanto riguarda il ME-OCSTP con istanze note in letturatura si vede che ottenere la soluzione ottima richiede un tempo computazionale elevato, mentre il MP-OCSTP è risolvibile in tempi più ragionevoli. Dopo aver valutato la qualità dei bound ottenuti con il rilassamento lineare, proporremo alcune disuguaglianze valide. Aggiungeremo quest'ultime risolvendo, quando necessario, un problema di separazione. I risultati computazionali

indicano che per il MP-OCSTP tutte le formulazioni portano a buoni risulati con un equivalente valore della funzione obiettivo quando le istanze soddisfano la disuguaglianza triangolare; mentre con istanze che non la soddisfano la formulazione con variabili a 4 indici risulta migliore. Per il ME-OCSTP la formulazione con variabili a 2 indice è più competitiva delle altre, nonostante i risultati siano meno soddisfacenti rispetto a quelli ottenuti per l'altro problema. Vedremo anche che in alcuni casi le disuguaglianze valide non migliorano i risultati ottenuti senza di esse.

Infine introdurremo alcune euristiche: le greedy saranno basate sul Minimum Spanning Tree, sulle stelle o sull'albero di Gomory-Hu, la ricerca-locale per il MP-OCSTP proverà a collegare i due nodi con il cammino più costoso in modo diretto, andando poi ad eliminare un arco nel ciclo che si genera e la ricerca locale per il ME-OCSTP proverà ad eliminare l'arco più costoso sostituendolo con un'altro che connetta le due componenti sconnesse. I risultati ottenuti mostrano che l'euristica basata sull'albero di Gomory-Hu, vista come il greedy più la ricerca locale, è la migliore per entrambi i problemi con molte istanze. A conclusione proveremo ad applicare proprio all'euristica basata sull'albero di Gomory-Hu una procedura randomizzata (GRASP), con la quale si otterrano soluzioni migliori.

Contents

Abstract iii

Sommario v

Contents vii

List of Figures ix

List of Tables xi

1 Introduction 1
- 1.1 Objectives . . . 1
- 1.2 Definition of the Problems . . . 2
- 1.3 Thesis Structure . . . 5

2 The Optimum Communication Spanning Tree Problem 7
- 2.1 Literature Review . . . 7
- 2.2 Applications . . . 9
  - 2.2.1 Alignment Problem in Computational Molecular Biology 9
  - 2.2.2 Tree of Hub Location Problem . . . 12

3 Notation and Preliminaries 15
- 3.1 Notation . . . 15
- 3.2 Preliminaries . . . 16
  - 3.2.1 The Gomory-Hu Tree . . . 16
  - 3.2.2 Linear Relaxation . . . 18
  - 3.2.3 Separation Problem . . . 18

4 Special Cases 21
- 4.1 Min-Max Path Optimum Distance Spanning Trees . . . 21
- 4.2 Min-Max Path Optimum Requirement Spanning Tree . . . 23

5 Mixed Integer Linear Formulations **27**
5.1 Formulation I with 4-Index Variables 27
5.2 Formulation II with 3-index Variables 29
5.2.1 Valid Inequalities 31
5.3 Formulation III with 2-index Variables 33
5.3.1 Valid Inequalities 36
5.4 Addition of Valid Inequalities 37

6 Heuristics **39**
6.1 Heuristic Algorithm 39
6.2 Greedy Algorithms 40
6.2.1 Minimum Spanning Tree 40
6.2.2 Heuristic with Costs Update 42
6.2.3 Star-Tree Heuristic 44
6.2.4 Heuristics based on Gomory-Hu Tree 44
6.3 Local-Search 45
6.3.1 Local-Search for MP-OCSTP 45
6.3.2 Local-Search for ME-OCSTP 46

7 Implementation and Results **49**
7.1 Analysis of existing Problems Instances 49
7.2 Implementation 50
7.3 Formulation I 50
7.3.1 Implementation 51
7.3.2 Computational Results 52
7.4 Formulation II 55
7.5 Formulation III 58
7.5.1 MP-OCSTP 58
7.5.2 ME-OCSTP 61
7.6 Comparison between the Three Formulations 62
7.6.1 MP-OCSTP 62
7.6.2 ME-OCSTP 64
7.7 Heuristic Algorithms for MP-OCSTP 66
7.8 Heuristic Algorithms for ME-OCSTP 70
7.9 Grasp Approach 74

8 Concluding Remarks **77**

Bibliography **79**

List of Figures

1.1 A Spanning Tree to connect some American cities. 2

2.1 Example of the alignment of three genomic sequences. 10
2.2 Example of the progressive alignment of genomic sequences. . . 11
2.3 Example of the Tree of Hubs Problem. 12

3.1 Graph example. 16
3.2 Cut definition example. 16
3.3 General idea of the Linear Programming Relaxation. 18
3.4 Flowchart of algorithm used to add cut-set inequalities. 19

4.1 Example for MDSTP 23
4.2 Example of min-max requirement spanning tree 24
4.3 Two solutions for the example 24

5.1 Explanation of relation (5.43)-(5.44) 34
5.2 Meaning of constraints (5.37) 34

6.1 Main idea of **HE-IV**. 42
6.2 Example of feasible solution for MP-OCSTP 45
6.3 Example of the edge that the Local-Search algorithm for MP-OCSTP tries to add. 46
6.4 Example of the edges that the Local-Search algorithm for MP-OCSTP tries to delete. 46
6.5 Example of feasible solution for ME-OCSTP 46
6.6 Example of the edges in the cut which separates i and j. 47
6.7 Example of the edges that the Local-Search algorithm for MP-OCSTP tries to add. 47

7.1 Comparison of computational times to solve MP-OCSTP . . . 62

List of Tables

2.1 Summary of some precedence works. 9

7.1 Results of formulation I for MP-OCSTP. 53
7.2 Results with formulation I for ME-OCSTP. 54
7.3 Results with formulation II for MP-OCSTP. 56
7.4 Results with formulation II for ME-OCSTP. 57
7.5 Results obtained using the formulation III for MP-OCSTP. . . 60
7.6 Results obtained using formulation III for ME-OCSTP. 61
7.7 Comparison LP relaxations of the three formulations for the MP-OCSTP. 64
7.8 Comparison LP relaxations of the three formulations for the ME-OCSTP. 65
7.9 Summary of Greedy Algorithms. 66
7.10 Results obtained using all the Greedy Algorithms to solve MP-OCSTP. 67
7.11 Deviation average for Greedy algorithms for the MP-OCSTP. . 68
7.12 Results obtained using the Local-Search Algorithm on the trees found with the Greedy Algorithms to solve MP-OCSTP. 69
7.13 Deviation average for Local-Search algorithm for the MP-OCSTP. 70
7.14 Results obtained using all the Greedy Algorithms to solve ME-OCSTP. 71
7.15 Results obtained using the Local-Search Algorithm on the trees found with the Greedy Algorithms to solve ME-OCSTP. 73
7.16 Result obtained with GRASP for the MP-OCSTP. 75
7.17 Result obtained with GRASP for the ME-OCSTP. 76

Chapter 1

Introduction

In recent years, considerable interest has been focused on various problems that arise in the area of network design. Such problems find applications in planning of transportation, communication system, water resource, distribution and computer networks.
One of such network design problems is known as the Optimum Communication Spanning Tree Problem. This problem was originally introduced by Hu in 1974 and in these years was studied by different authors. It is characterized by a set of users which need to communicate with each other, because there is a demand of communication between each pairs of them. Every link that connects a pair of users has a cost for every unit of communication that uses this link. This cost can be interpreted for instance as the distance between the users. The objective of this problem is to build a spanning tree connecting these users such that the Total Cost of Communication of the spanning tree is minimum among all spanning trees. Remember that a spanning tree is a network that connects all the nodes and contains no cycles. Figure 1.1 shows an example of the situation explained above.

1.1 Objectives

In this work we study two possible variants of the Optimum Communication Spanning Tree Problem. Suppose, for example, that users correspond to cities and communication to telephone calls between pairs of cities. Then, when the telephone calls of each pair of cities are supervised by different phone companies, each company wants to minimize its costs. In this case, the objective of the problem becomes to build the spanning tree such that the cost of the

Figure 1.1: A Spanning Tree to connect some American cities.

most expensive path between a pair of cities is minimum. Moreover, we can interpret the cost of a direct link between two cities in a different way: it is the money loss for every call which cannot be taken place in the event that the connection is broken. Then, the total cost of a connection represents the money loss due to the total calls failed and in this case we want to build the spanning tree with the minimum cost of the most expensive connection.
To solve these two variations of the problem we propose Mixed Integer Linear Programming (MILP) formulations and some heuristic algorithms. Thanks to that we can compute lower bounds and upper bounds of the optimal value. We run computational experiments using Euclidean and randomly generated distances/costs.
First, we dedicate the following Section to understand more clearly these problems through a mathematical description.

1.2 Definition of the Problems

Optimum Communication Spanning Tree Problems are formally defined as follows. Let G=(V, E) be a complete undirected graph with

- $n = |V|$ nodes and $m = |E|$ edges (as G is complete $m = n(n-1)/2$).
- A Cost function over the edges (normally associated with the length the

of edges),

$$c : E \to \mathbb{R}^+$$
$$e \mapsto c_e.$$

The costs will be represented with a Cost Matrix:

$$\begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix}$$

We make the following three assumptions: all the costs are non-negative ($c_{ij} \geq 0$), symmetric ($c_{ij} = c_{ji}$) and $c_{ii} = 0$ for all $i \in V$.

- A Communication Requirement function between pairs of nodes:

$$r : V \times V \to \mathbb{R}^+ \cup \{0\}$$
$$(i, j) \mapsto r_{ij},$$

that is represented with the Communication Requests Matrix:

$$\begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{pmatrix},$$

where $r_{ij} \geq 0$. In particular r_{ij} is equal to 0 if $i = j$. Since the cost function is symmetric, for the sake of simplicity, we consider this matrix upper triangular:

$$\begin{pmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ r_{21} & r_{22} & \dots & r_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n1} & r_{n2} & \dots & r_{nn} \end{pmatrix} \longrightarrow \begin{pmatrix} 0 & r_{12} + r_{21} & \dots & r_{1n} + r_{n1} \\ 0 & 0 & \dots & r_{2n} + r_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix}$$

Henceforward, we redefine r_{ij} as $r_{ij} + r_{ji}$ for all $i, j \in V, i < j$.

The ***Total Communication Cost of T*** is the sum of the communication costs over T of all pairs of nodes.

$$CC(T) = \sum_{(i,j) \in V \times V : i < j} CC_{ij} = \sum_{(i,j) \in V \times V : i < j} r_{ij} c^T(i, j).$$

The **OCSTP** is to find a spanning tree of G of minimum total communication cost.

The ***Communication Cost over a tree T of a pair of nodes*** $i, j \in V, i \neq j$ is defined as the communication requirement between the pair multiplied by the cost of the (unique) path in T that connects them:

$$CC_{ij}^T = r_{ij} c^T(i, j).$$

The Minimum Path Optimum Communication Spanning Tree Problem (**MP-OCSTP**) is to find a spanning tree of G where the maximum communication cost between all pairs of nodes is minimum.
For a given tree T, let f_{ij}^T denote the total amount of flow that circulates through any of two directions of (i, j). f_{ij}^T is the sum of the communication requirements of all pairs that use or (i, j) or (j, i) in their path.
Then, the ***Communication Cost over T of an edge of T*** is given by the flow multiplied by the cost of the edge:

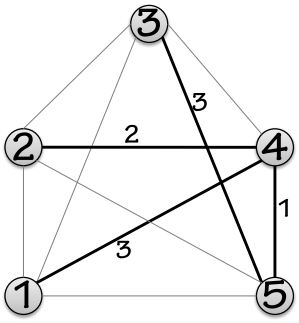
$$FF_{ij}^T = c_{ij} \cdot f_{ij}^T.$$

The Minimum Edge Optimum Communication Spanning Tree Problem (**ME-OCSTP**) is to find a spanning tree of G where the maximum communication cost of any edge is minimum.

Now, we present an example in order to clarify how calculate all types of communication costs defined above.

Consider the following spanning tree T with requirement matrix R. So, we can compute in the tree the costs c^T of each path which connect a pair of nodes.

$$R = \begin{pmatrix} 0 & 3 & 5 & 7 & 2 \\ 0 & 0 & 8 & 1 & 4 \\ 0 & 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



$$c^T = \begin{pmatrix} 0 & 5 & 7 & 3 & 4 \\ 5 & 0 & 6 & 2 & 3 \\ 7 & 6 & 0 & 4 & 3 \\ 3 & 2 & 4 & 0 & 1 \\ 4 & 3 & 3 & 1 & 0 \end{pmatrix}$$

First, we calculate *Communication Cost over the tree T of all pair of nodes*:

$$CC_{12}^T = r_{12} \cdot c_{12}^T = 3 \cdot 5 = 15 \qquad CC_{24}^T = r_{24} \cdot c_{24}^T = 1 \cdot 2 = 2$$
$$CC_{13}^T = r_{13} \cdot c_{13}^T = 5 \cdot 7 = 35 \qquad CC_{25}^T = r_{25} \cdot c_{25}^T = 4 \cdot 3 = 12$$
$$CC_{14}^T = r_{23} \cdot c_{14}^T = 7 \cdot 3 = 21 \qquad CC_{34}^T = r_{34} \cdot c_{34}^T = 2 \cdot 4 = 8$$
$$CC_{15}^T = r_{23} \cdot c_{15}^T = 2 \cdot 4 = 8 \qquad CC_{35}^T = r_{35} \cdot c_{35}^T = 3 \cdot 3 = 9$$
$$\underbrace{CC_{23}^T = r_{23} \cdot c_{23}^T = 8 \cdot 6 = 48 \qquad CC_{45}^T = r_{45} \cdot c_{45}^T = 6 \cdot 1 = 6}$$

Objective function value of MP-OCSTP = 48

Then, we can calculate the *Total Communication Cost of T*, which is also the objective function value of the OCSTP:

$$CC(T) = 15 + 35 + 21 + 8 + 48 + 2 + 12 + 8 + 9 + 6 = 164.$$

Finally, to calculate the *Communication Cost over T of all edges of T*, we consider:

$$\begin{array}{ll} \text{Flow through } (1,4) = r_{14} + r_{12} + r_{13} + r_{15} = 17 & FF_{14}^T = 3 \cdot 17 = 51 \\ \text{Flow through } (2,4) = r_{24} + r_{12} + r_{23} + r_{25} = 16 & FF_{24}^T = 2 \cdot 16 = 32 \\ \text{Flow through } (3,5) = r_{35} + r_{13} + r_{12} + r_{34} = 18 \quad \Longrightarrow & FF_{35}^T = 3 \cdot 18 = 54 \\ \text{Flow through } (4,5) = r_{45} + r_{34} + r_{13} + r_{23} + r_{25} + r_{15} = 27 & \underbrace{FF_{45}^T = 1 \cdot 27 = 27} \end{array}$$

Objective function value of ME-OCSTP = 54

1.3 Thesis Structure

In **Chapter 2** we review the Optimum Communication Spanning Tree Problem literature. Moreover we present two specific applications of this problem. The first is one in the field of the computational biology, since this problem can be exploited to produce a heuristic algorithm to solve the alignment of genomic sequences. The second one is an application in the hub location area.
In **Chapter 3** we introduce some notations and preliminary kwll-known results, that we use during this work.
In **Chapter 4** we study the solution properties of two particular cases. In fact, we initially suppose that all the communication requirements are the same, then we suppose that all costs are the same.
In **Chapter 5** we propose three different formulations to solve the MP-OCSTP and the ME-OCSTP. These formulations differ from each others in the number of variables that they use. Moreover, we present some valid inequalities which can reinforce the formulations.

In **Chapter 6** we present some heuristic algorithms. In particular, we introduce greedy algorithms which produce a spanning tree that is an admissible solution for both problems. Then, for each problem we propose a local-search algorithm to improve the results obtained with the previous algorithm.
In **Chapter 7** we explain how the formulations have been implemented using a specific program; and we present the computational results obtained trying to find out the formulation which produces the best results. We conclude this chapter presenting the outcome produced by all the heuristic algorithms.
Finally, in **Chapter 8** we summarize this thesis.

Chapter 2

The Optimum Communication Spanning Tree Problem

Most previous works in literature are focused on the original problem where the total communication costs must be minimized. Thus, we dedicate this Chapter to recall some works and results. Then, we present in Section 2.2 two specific applications of the problem.

2.1 Literature Review

The OCSTP was originally introduced in 1974 by Hu [11]. The difficulty of the general case of the OCSTP motivated that he focused on two particular cases that can be solved in polynomial time. The first one is the Optimum Distance Spanning Tree (ODSTP), where communication requirement between all pairs of nodes is the same. In this case, there is an optimal solution which has a star topology, under some additional condition. The second particular case is the Optimum Requirement Spanning Tree Problem (ORSTP) where all pairs of nodes have the same cost/distance. Now, an optimal solution is given by a Gomory-Hu Tree.
A few year later, Johnson, Lenstra and Rinnooy Kan [12] showed that the OCSTP is $\mathcal{NP}$-hard.
In 1987 Ahuja and Murty [1] developed an exact algorithm based on Branch and Bound (B&B) and a heuristic algorithm for solving the problem. The B&B algorithm uses the *lower plane* to find a valid lower bound at a every vertex of the enumeration tree. A lower plane is a linear lower approximation of the value of objective function, obtained considering all the nodes of the graph, but

only a subset of the arcs. The complexity of this procedure is $O(n^4)$ and the performance of the whole algorithm depends on the type of data. Nevertheless, the authors were able to prove the optimality of the solutions for instances with up to 40 nodes. The heuristic algorithm consists of two phases: tree-building and tree-improvement. The idea for the construction of the tree is derived from the well-known Minimum Spanning Tree (MST), while the improvement phase examines each arc in the current tree and its possible exchange with another one outside the current tree. Despite its complexity ($O(n^3)$), this algorithm produces excellent solutions. For example, it was capable of finding optimal solutions with 100 nodes and 1000 arc in very short times.

Later, in [14] the author compares the features of optimal solutions for OCSTP and the features of trees generated randomly. He showed that, on average, the difference between optimal solutions and minimum spanning trees is smaller than the difference between optimal solutions and randomly trees. Moreover, if the distances/costs of the graph are randomly generated, the MST produces values closer to the optimal solution than if the problem has Euclidean distances/costs. Thus, this similarity between MSTs and optimal values suggested that the performance of some heuristics for the OCSTP could improve by starting with an MST. this strategy allowed to obtain the same solutions but 10 times faster than when a random tree was used initially. To the best of our knowledge, there is no exact algorithm which solves size OCSTP instances. Therefore line of research is to find a successful formulation using linear programming (LP). Contreras [2] proposed an integer linear programming formulation which, with some valid inequalities, produced optimal solution for instances with up to 25 nodes in reasonable time. Then Contreras, Fernández, Marin [3] presented a Lagrangian relaxation which produced a good lower and upper bound for instances with up to 50 nodes. In this formulation, they used variable with 4 index. The disadvantage of such formulation is the impossibility to solve instances with up to 30 nodes with a general solver. On the other hand, it is possible to formulate this problem using variables with only 2 index [7]. Although it is compact, it produced very weak lower bounds so it required too much computing time, even for very small instances.

In [8] the authors proposed a compromise with a formulation where the variables have 3 indeces. This formulation could not solve instances of moderated sizes, but the authors reinforced it with some valid inequalities. Depending on the instance, the obtained lower bound was a 80-95% of the optimal value. Thus, a promising avenue of research is to develop a new families of valid inequalities and new mathematical formulations for the OCSTP.

Year	Author	Purpose&Result
1974	T.C.Hu	Study of two special cases.
1978	Johnson, Lenstra and Rinnooy Kan	Proven NP-Hard.
1987	Ahuya and Murty	Exact method and some heuristics.
2009	Rothlauf	Comparison between optimal solutions and the features of tree generated randomly.
2009	Contreras	Integer linear programming formulation.
2010	Contreras, Fernandez and Marin	Lagrangian relaxation (4-index variables).
2012	Fernández, Luna-Mota and Reinelt	Formulation and valid inequalities with 2-index variables.
2013	Fernández, Luna-Mota, Hildenbrandt, Reinelt and Weisberg	Formulation and valid inequalities with 3-index variables.

Table 2.1: Summary of some precedence works.

2.2 Applications

Besides classical applications in telecommunications, we have already mentioned, the OCSTP can also be applied to other fields. Now, we see in detail two types of applications, one in the field of the computational biology, the second one in hub location.

2.2.1 Alignment Problem in Computational Molecular Biology

A broadly studied problem in molecular biology is called *multiple sequence alignment*. For recognizing evolutionary relationships, for identifying regions of preserved DNA and for finding fatal mutations, the biologists compare genomic sequences drawn by individuals of the same or different species. In recent years they have focused on the search of a mathematical formulation and its resolution. In particular, it is known that the above problem can be formalized as an optimization problem. In [9] the authors exploit the Optimum Communication Spanning Tree to produce a heuristic for this problem. First

of all, to understand the connection between these two problems it is necessary analyze the multiple sequence alignment problem.
A genomic sequence consists of a string composed of 4 alphabetical symbols of nucleotides or 20 alphabetical symbols of amino acids. The sequences are inserted in a matrix, each row of which is occupied by a sequence. With the addition of gaps all the sequences have the same length and, in every column, the same character always appears. The alignment corresponding to the sequences ATTCGAC, TTCCGTC, and ATCGTC is depicted in Figure 2.1. The purpose is to identify some common patterns.

Typically, the objective function in the multiple alignment problem is a gen-

A T T - C G A - C
- T T C C G - T C
A - T - C G - T C

Figure 2.1: Example of the alignment of three genomic sequences.

eralization of the alignment of two sequences (*pairwise alignment*). Then, c(a, b) denotes the cost for replacing the character a with the character b and viceversa, and $c(-, a)$ the cost for deleting/inserting of character a. The problem is to find minimum cost operations that allow to transform the sequence S' in S''. The optimal value is called *edit distance* and it is denoted by $d(S', S'')$. An alignment $\mathcal{A}$ of two or more sequences is an array having the (gapped) sequences as rows, and it can be seen as a path of replacements and/or insertions. The value $d_{\mathcal{A}}(S', S'')$, where S' and S'' are two sequences, is obtained adding up the cost to align S' and S'' and $d(S', S'') = \min_{\mathcal{A}} d_{\mathcal{A}}(S', S'')$. Therefore an alignment of n sequences (*SP-alignment*) is obtained by adding the costs of the pairs of the symbols matched up at the same positions, that is $SP(T) = \sum_{(S',S'')} d_{\mathcal{A}}(S', S'')$.
Obtaining a solution with dynamic programming to solve the above problem takes a time proportional to $2^n l^n$, where l is the maximum length that a sequence can assume. Considering that in real life problems n is a small number and l is of the order of several hundreds, dynamic programming is able to produce solutions for small instances. In [16] the authors say that some bound criteria have been introduced in order to reduce the time and the space requirements of dynamic programming and make solvable problems for $n \leq 6$ and $l \leq 200$. In every case, constructing optimal alignment is computational expensive, since the problem has been shown to be NP-complete (Wang and Jiang, [15]).

In the same article the authors explain for the SP-alignment the Gusfiled'algorithm which uses an approach to multiple alignment guided by a tree, due to Feng and Dolittle [6]. Other algorithms and heuristics based on the Feng and Dolittle progressive approach have been proposed: an alignment is built by considering one sequence at a time. This develops in two phases:

1. Find a heuristic tree;
2. Use the tree as a guide for aligning sequences iteratively, as described after.

Given a tree T, where every sequence is a node, then exists a multiple alignment $\mathcal{A}(T)$ such that $d_{\mathcal{A}(T)}(S', S'') = d(S', S'')$ is verified for all the pairs (S', S'') connected from an edge of T. We can obtain a final alignment as follows (as the Figure 2.2): (i) Pick an edge (the cut) and align recursively the sequences on both sides of the cut; (ii) align optimally the two sequences to the endpoints of the edge, whose cost coincides with the edit distance; and (iii) use this alignment to merge all the sub-alignments in one only (columns of gaps are inserted where the pairwise alignment sets a gap in one of the two sequence).

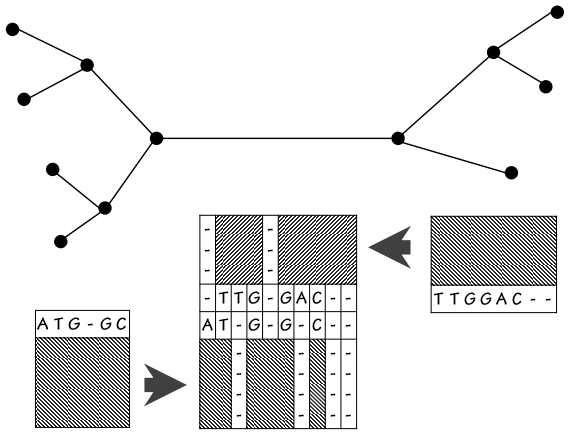


Figure 2.2: Example of the progressive alignment of genomic sequences.

A tree that can be used to minimize the total pairwise distance in the multiple alignment is the Minimum Optimum Communication Spanning Tree. In [9] they propose a heuristic for multiple alignment based on the Optimum Communication Spanning Tree built on the graph, where the nodes are the sequences and the edges are weighed using the edit distance. This heuristic has been tested on several families of proteins, and the results show a reduction of the 10% of the costs in comparison to other algorithms previously used. Moreover

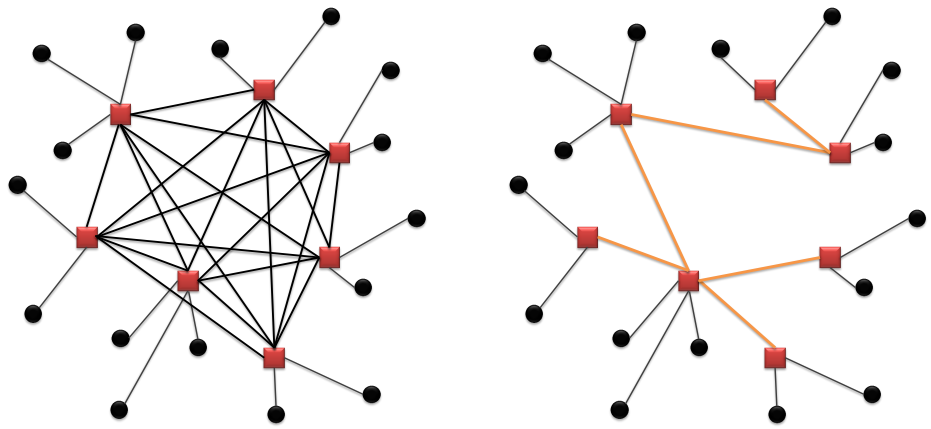
the heuristic leads to a value that is within 8% of the optimum, on average.

2.2.2 Tree of Hub Location Problem

This application shows that the OCSTP is a sub-problem of the tree of hub location problem. It is known that hub location is a very challenging area due to the economic impact of potential applications. Hub location problems stem from transportation and telecommunication systems, where several origin/destination points send and receive some product. Normally, there are hubs, set of points which are used to redistribute the flow and to reduce transportation costs. In may hub location models there is an optimal solution in which hubs are fully interconnected, namely there is a link connecting any pair of hubs. There exist, however many application in which this is not true. An example of this is the tree hub location problem (THLP), where the hubs are connected by a tree.

The THLP is defined on a directed graph, where a flow must be send though the network between each couple of nodes. For this, p hubs have to be located and connected by a tree. Each node will be allocated to only one hub and all the flow from/to the node has to pass through its allocated hub. There is a unit transportation cost associated with each arc. The objective is to minimize the operation cost of the system.

This is a problem where location, design and routing decisions have to be



(a) *Example of one possible instance.* (b) *A possible solution for the instance.*

Figure 2.3: Example of the Tree of Hubs Problem.

taken: where to locate the hubs, how to interconnect the selected hubs, how to

allocate each node to a single hub and how to route the flows between vertices of the network. If the location of the hubs and the allocation of non-hubs to hubs are given, this problem becomes an OCSTP (Figure 2.3). For this reason, a procedure to solve efficiently the OSCTP may also serve as a routine for solution procedures for the THLP.

Chapter 3

Notation and Preliminaries

In this chapter, we introduce the notation which is used in this work. Then, in Section 3.2 we present some preliminary concepts that we use during all the work.

3.1 Notation

Let S be a subset of V. We define:

- $E(S) = \{e = (i, j) \in E : i, j \in S)\}$, all the edges with its two end-nodes in S.
- $\delta(S) = \{e = (i, j) \in E : (i \in S, j \notin S)$ or $(i \notin S, j \in S)\}$, all the edges with one end-node in S and the other one outside S.

When S is a singleton, i.e. $S = i$:

- $\delta(i) = \{e \in E : e = (i, j)$ or $e = (j, i)\}$, all the edges where one end-node is i. $\delta(i)$ is called *degree* of node i.

If we consider $G' = (V, A)$, the directed graph obtained by defining, for each edge in E, two arcs, one in each direction, we can define:

- $A(S) = \{a = (i, j \in E : i, j \in S)\}$, all the arcs with both end-nodes in S.
- $\delta^-(S) = \{a = (i, j) \in A | i \notin S, j \in S\}$, all the arcs outgoing from S.
- $\delta^+(S) = \{a = (i, j) \in A | i \notin S, j \in S\}$, all the arcs incoming in S.
- $\delta^-(i) = \{a = (i, j) \in A\}$, all the arcs outgoing from i.
- $\delta^+(i) = \{a = (j, i) \in A\}$, all the arcs incoming i.

3.2 Preliminaries

3.2.1 The Gomory-Hu Tree

Before presenting the Gomory-Hu Tree, we give some definitions.

Consider an undirected graph $G' = (V', E')$ with edges capacities b_{ij} for all $(i, j) \in E'$.

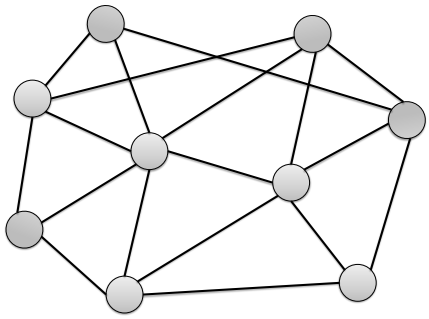


Figure 3.1: Graph example.

Definition 3.1. A *cut*, denoted by $(X, \bar{X})$, where X is a subset of V and $\bar{X}$ is its complement, is the set of edges with one end-node in X and the other on in $\bar{X}$.

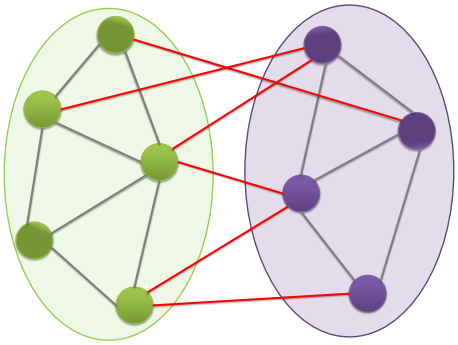


Figure 3.2: Cut definition example.

The capacity of a cut, denoted by $b(X, \bar{X})$, is the sum of the capacities of all edges in the cut, i.e.

$$b(X, \bar{X}) = \sum_{(i,j):i \in X, j \in \bar{X}} b_{ij}$$

Moreover, let fl_{pq} be the value of the maximum flow from p to q in G'.
By the max-flow min-cut theorem, there always exists a cut $(X, \bar{X})$, with $p \in X$

and $q \in \bar{X}$ such that $b(X, \bar{X}) = fl_{pq}$. The cut $(X, \bar{X})$ is called *minimum cut*.

To compute the capacity of the minimum cut for a pair of nodes (s, t), we can use the well-known Algorithm of Ford and Fulkerson. Since the complexity of this algorithm is proportional to m, to compute the value of minimum cut of all pairs of nodes of the graph it is better to find out the Gomory-Hu Tree ([10]).
It is a tree with the following properties:

(i) Each edge of the spanning tree has a value v_{ij} associated with it. If we delete the edge with value v_{ij}, the network is disconnected into two components, X and $\bar{X}$. Then, $v_{ij} = b(X, \bar{X})$ and $(X, \bar{X})$ is a minimum cut of G'.

(ii) The maximum flow f_{pq} between two nodes p and q is

$$f_{pq} = min(v_{pu}, ..., v_{ij}, ..., v_{uq}),$$

where $v_{pu}, ..., v_{ij}, ..., v_{vq}$ are values associated with edges which form the unique path connecting p and q in the tree.

To find the Gomory-Hu Tree we use the Gusfield's Algorithm ([10]) which builds a rooted directed tree.
Let p be a vector with length n, where we save in position i the predecessor of i. It is initialized to 1. The edges of T are the final pairs $(i, p[i])$ for all i from 2 to n, and edge $(i, p[i])$ has value $fl(i)$. If each edge is thought as a directed edge from i to $p[i]$, then T forms a directed tree rooted to node 1.

Algorithm 3.1: Gusfield's Algorithm

```
1  input: G=(V, E)
2  begin
3     p=1;
4     for s:=2 to n
5        t:=p[s]
6        (X X̄ , f(s,t))= Ford&Forkerson (G, s, t)
7        let X be the set of nodes on the s side of the cut
8        fl[s]:= f(s,t);
9        for i:=1 to n
10          if i ≠ and i is in X and p[i]=t then p[i]:=s;
11       end
12       if p[t] is in X
13       then
14          p[s]:=p[t];
15          p[t]:=s;
16          fl[s]:=fl[t];
17          fl[t]:=f[s,t];
18       end
19    end
```

```
20      return (p, fl);
21  end
```

3.2.2 Linear Relaxation

In general, to solve a linear programming relaxation of a minimization problem the integrality constraints on the variables are removed, as shown in Figure 3.3, and the domain of integer variables becomes the whole continuous interval.

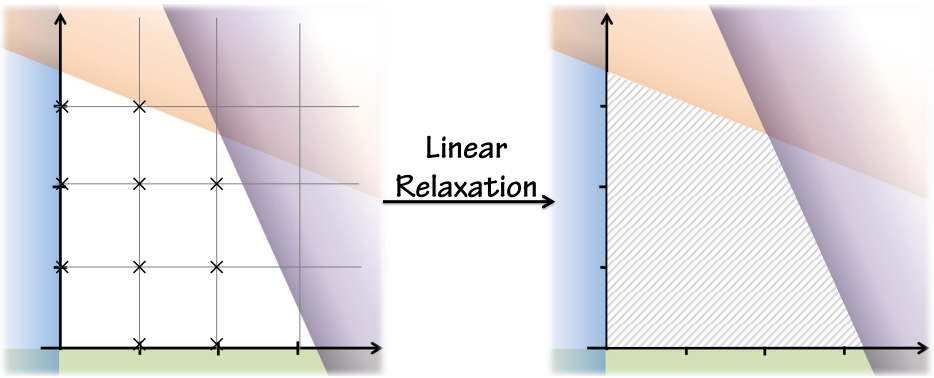


Figure 3.3: General idea of the Linear Programming Relaxation.

This means that a linear integer program (LIP) becomes linear programming (LP) problem. Since we have removed the integrality constraints, we have extended the domain of the problem and the optimal value $Z(T')$ is such that

$$Z(T') \leq Z(T^*),$$

where $Z(T^*)$ is the optimal value of the original (unrelaxed) problem. Therefore the optimal value obtained solving the continuous relaxation represents a lower bound of $Z(T^*)$.
When we will present the results, to evaluate the quality of the lower bounds we will calculate:

$$\frac{Z(T')}{Z(T^*)}.$$

This is a percentage value. In fact, if the relaxation produces the optimal value, we can say that the lower bound is the 100% of the optimal value. For this reason, the quality of a lower bound increases with this value.

3.2.3 Separation Problem

Now, we present the general idea of the algorithm used for adding exponential family of inequalities, which is also showed in Figure 3.4. At each iteration the

following steps are performed:

Step 1 Solving a relaxation of the problem.

Step 2 Solving the *separation problem* for finding inequalities violated by the optimal solution of the current relaxation.

Step 3 Adding the violated inequalities to the current system.

This procedure can be used to add valid inequalities to reinforce a formulation or to add a constraint. In both cases, it is used only if the number of inequalities is exponential.

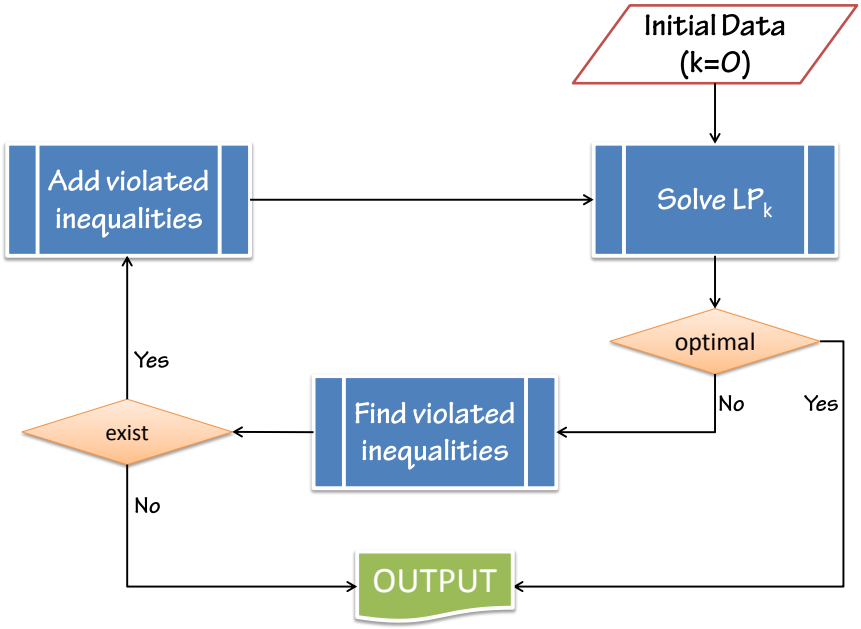


Figure 3.4: Flowchart of algorithm used to add cut-set inequalities.

Chapter 4

Special Cases

In this chapter , we focus ourselves on two special cases of MP-OCSTP and we discuss some results about their solution.

case A The costs c_{ij} are arbitrary, while the requirements r_{ij} are all equal. This case is referred to as the *min-max path optimum distance spanning tree problem* (MP-ODSTP).

case B The costs c_{ij} are all equal, while the requirements r_{ij} are arbitrary. This case is referred to as the *min-max path optimum requirement spanning tree problem* (MP-ORSTP).

4.1 Min-Max Path Optimum Distance Spanning Trees

Without loss of generality, we can assume that in this case $r_{ij} \equiv 1$ and c_{ij} arbitrary. Throughout this section, we assume that there $n \geq 4$. First we give some definitions and notions. Given a tree T, we define:

Definition 4.1. A node of T is an *outer node* if its degree is one.

Definition 4.2. A node of T is a *inner node* if its degree is two or more.

Definition 4.3. A tree is called a *star-tree* if there is only one inner node in the tree.

Definition 4.4. A inner node of a tree is called *extreme* if, in the case we erase all outer nodes from the tree, it becomes an outer node.

Whether the goal is to minimize the total cost or to minimize the cost of the more expensive path, in general an optimum spanning tree may not to be a star-tree. For this reason, we want to define a sufficient condition hat guarantees that there is an optimal solution that is a star-tree.

Lemma 4.1. *If all the c_{ij} are equal, then there is an optimal solution to the MP-ODSTP which is a star-tree.*

Proof. Suppose $c_{ij} = c \neq 0$ for all $(i, j) \in E$. Let T denote a given spanning tree. If T is a star-tree, let s the only inner node of T. Then there are two possible types of paths:

- s is or the origin or the destination of the path. In this case, the cost is c;
- The path connects two nodes, i and j, different from s. In this case, the cost of the path is $c_{is} + c_{sj} = 2c$.

Therefore the cost of the most expensive path is $2c$.
If the spanning tree is not a star-tree, then there are at least two inner nodes: s and s'. If we consider i and j, two outer nodes that are respectively connected to s and s', the cost of the path $i - j$ is equal to $c_{is} + c_{ss'} + c_{s'j} = 3s$. Since $3c \geq 2c$, the *optimum distance spanning tree* is a *star-tree.* □

What we would want to find is a sufficient condition to conclude that there is a solution for MP-ODSTP that is a star-tree for the case when all costs are not necessarily the same. For the Optimum Distance Spanning Tree, where the total cost is minimized, the sufficient condition requires that all the costs of the edges "do not differ too much", as stated formally in the following theorem:

Theorem 4.1. *[11] Consider the MDSTP. Let a, b and c be the costs of three sides of any triangles in the n-node network ($n \geq 4$), where*

$$a \leq b \leq c. \tag{4.1}$$

If there exists a positive t not larger than $(n-2)/(2n-2)$ such that

$$a + tb \geq c \tag{4.2}$$

for all triangles in the network, then there exists an optimum distance spanning tree which is a star-tree.

In [11] Hu gives a proof for this theorem, which we do not reproduce here. We will exploit, however, his proof to show that the theorem is not valid for the MP-ODSTP.

If we consider the simple example in Figure 4.1, we show that this affirmation is not true for MP-ODSTP. A spanning tree T with at least two inner nodes is considered. Let q be an extreme inner node in T linked with p, which is an inner node.

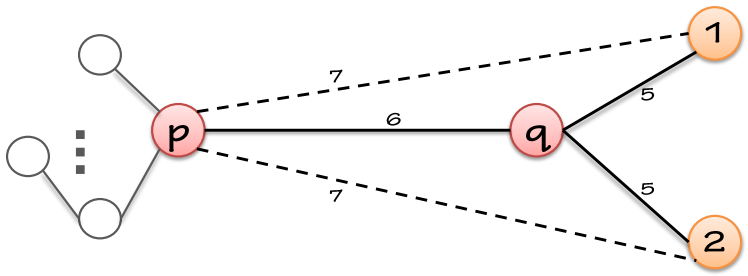


Figure 4.1: Example for MDSTP

In the example we have that $a \leq b \leq c$ for each of the 2 triangles. Moreover $n \geq 5$ implies $(n-2)/(2n-2) \geq 3/8$. If we take t=3/8, the condition $a+tb \geq c$ holds for both triangles. We have shown that the hypothesis of Theorem 4.1 hold. The cost of the path $1-2$ is 10, but if we erase the edge $p-q$ to convert the tree in a star-tree, the cost becomes $7+7=14$, which is greater than the most expensive path in the solution drawn. So we have shown that this theorem is not valid for our problem.

4.2 Min-Max Path Optimum Requirement Spanning Tree

In this section, we assume the costs c_{ij} all equal to $c \neq 0$, while the requirements r_{ij} are arbitrary.
For this second case, Hu shows that for Optimum Requirement Spanning Tree, where the total cost is minimized, the following theorem holds:

Theorem 4.2. *The cut-tree is an optimum requirement spanning tree.*

In the theorem cut-tree refers to the tree of minimum-cuts taking the requirements as capacities. It is well-known that this tree can be obtained with the algorithm of Gomory-Hu. It is easy to see that in our case, the theorem is not valid. This is illustrated by the example of Figure 4.2, which depicts the data of an instance for the MP-ORSTP. An edge connecting two vertices i and j indicates that that requirement between i and j, r_{ij}, is strictly positive. In this case, the value r_{ij} is written next to the edge. Therefore the edges not represented have a requirement equal to zero.

In Figure 4.3 two spanning trees are represented. Figure 4.3a represents the tree obtained with the algorithm of Gomory-Hu (T_1), while Figure 4.3b shows the optimal solution for the instance (T_2), with a value of the objective function

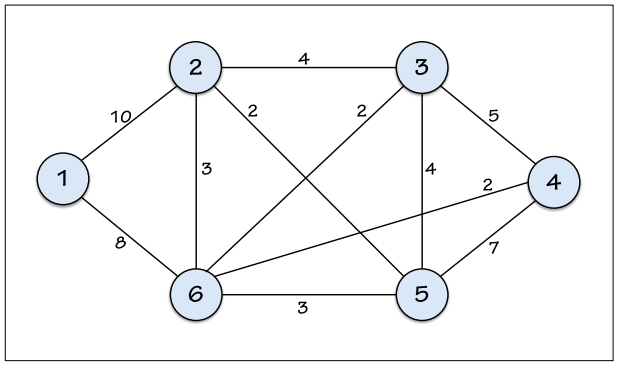
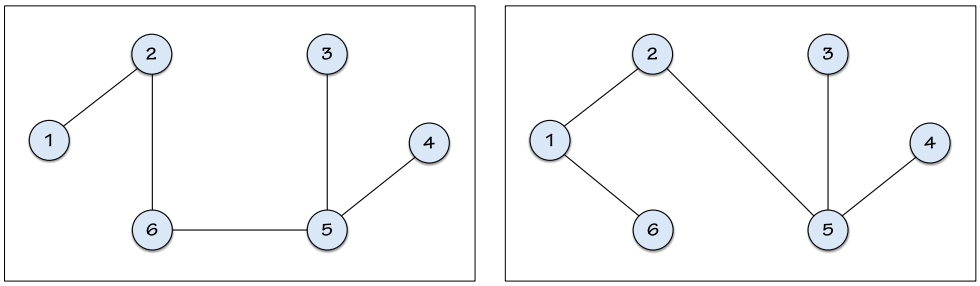


Figure 4.2: Example of min-max requirement spanning tree



(a) T_1: *Cut-tree for the example.* (b) T_2:*Optimal solution for the example.*

Figure 4.3: Two solutions for the example

equal to 10. For example, if we compute $CC_{23}^{T_1}$, it is equal to 4 (the requirement) multiplied by 3 (the number of edges that connects them). Thus in T_1 a path with cost 24 exists, which is greater than the optimal value. With this simple example we have shown that, with a different objective function, the theorem does not remain valid.

From Lemma 4.1, we already know that if the requirements are all equal, then the solution is a star-tree. As we see below, it is not necessary that all requirements are equal, but the statement remains true imposing a less restricting condition.
First of all we observe that when all edge costs are equal, the cost of a path reduces to the requirement multiplied for the number of edges in the path multiplied by the cost c of the edges. Therefore, roughly speaking, to guarantee that there is an optimal solution which is a star tree, it is enough that all requirements are similar except one which is much larger.

Lemma 4.2. *In the min-max requirement spanning tree problem, let R the*

largest requirement and suppose that for all $r_{ij} \neq R$ it holds that

$$r_{ij} \leq \frac{R}{2}.$$

Then, there exists an optimum requirement spanning tree which is a star-tree.

Proof. Firstly, we observe that the value cR is a lower bound of the solution. In fact, connecting the two nodes having requirement equal to R costs at least cR.
Let T denote a star-tree, where s is the only inner node and is one of the two nodes, i or j, which have R as requirement. The cost of the path (i, j) is cR. Moreover, there are another two possible types of paths:

- s is or the origin or the destination of the path. In this case, the cost is lesser than $c\frac{R}{2}$;
- The path connects two nodes, u and v, different from s. In this case, the cost of the path is $cr_{is} + cr_{sj} < cR$.

Therefore the cost of the most expensive path is equal to cR, which is also a lower bound of the problem. So, there is a solution of the MP-ORSTP that is a star-tree. □

We observe that it is not necessary that only one pair of nodes has a "big" requirement. In fact, there may be other pairs with "big" requirement; but, all of them must have the same node in common.

Chapter 5

Mixed Integer Linear Formulations

In this chapter, we propose three different formulations for MP-OCSTP and for ME-OCSTP. These formulations differ from each other in the number of index on the variables chosen. In particular, in Section 5.1 we present a formulation with 4-index variables, in Section 5.2 with 3-index variables and in Section 5.3 with 2-index variables. For each of these formulations we propose some families valid inequalities.

5.1 Formulation I with 4-Index Variables

In the first formulation for MP-OCSTP and ME-OCSTP we use four index variables, using for routing flows in the solution tree, which are related to 2-index design variables.
For every edge in the graph, we define the following:

- x_{ij}, a binary decision variable which is equal to 1 if and only if edge (i, j) is in the tree. It is defined for all $i, j \in V, i < j$.

In addition, associated with each origin/destination pair, we define the following set of continuous variables to represent the arcs of a path connecting the origin/destination pair:

- y_{ij}^{uv}, a binary decision variable which indicates if the directed arc (i, j) is on the path from u to v. It is defined for all $i, j, u, v \in V, u < v$.

Then, a valid formulation is:

Set of constraints and variable domains.

$$\sum_{j \in V \setminus \{u\}} y_{uj}^{uv} = 1 \qquad \forall u, v \in V, u < v, \forall r_{uv} > 0 \tag{5.1}$$

$$\sum_{i \in V \setminus \{j,v\}} y_{ij}^{uv} - \sum_{k \in V \setminus \{j,u\}} y_{jk}^{uv} = 0 \qquad \forall u, v, j \in V, u < v; \forall j \neq u, v, \forall r_{uv} > 0 \tag{5.2}$$

$$y_{ij}^{uv} + y_{ji}^{uv} \leq x_{ij} \qquad \forall u, v, i, j \in V, i < j, u < v, \forall r_{uv} > 0 \tag{5.3}$$

$$\sum_{i,j \in V; i<j} x_{ij} = n - 1 \tag{5.4}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i < j \tag{5.5}$$

$$y_{ij}^{uv} \in \{0, 1\} \qquad \forall u, v, i, j \in V, u < v \tag{5.6}$$

Constraints (5.1)-(5.2) guarantee that there is a path connecting the endnodes of each origin/destination pair. The family of constraints (5.1) guarantees that, in the path connecting u and v, there is only one active arc outgoing from node u. For each $u, v \in V$, constraints (5.2) ensure that each node $j \in V$ has the same number of incoming and outgoing arcs. Constraints (5.3) relate the y and x variables by activating the edges that are used for at least one path (u, v). Constraint (5.4) guarantees that the solution is a spanning tree, by limiting the total number of edges to $n - 1$. Finally, variable domains (5.5) and (5.6) complete the model.

In this model, there are $mn(n-1) = \mathcal{O}(mn^2)$ variables and $1 + m + m^2 + m(n-1) = \mathcal{O}(m^2)$ constraints.

Objective function for MP-OCSTP.

$$\min \max_{u,v \in V; r_{uv} > 0} r_{uv} \sum_{i,j \in V; i \neq j} c_{ij} y_{ij}^{uv} \tag{5.7}$$

Multiplying the requirement by the cost of the unique path connecting two nodes, we have the total communication cost for that pair of nodes. Thus, we want to minimize the cost of the more expensive path.

Objective function for ME-OCSTP.

$$\min \max_{i,j \in V; i<j} c_{ij} \sum_{u,v \in V;} r_{uv}(y_{ij}^{uv} + y_{ji}^{uv}) \tag{5.8}$$

Multiplying the cost of the edge by the requirements of all pairs of node that use the edge for transporting the flow, we have the total communication cost for that edge. Thus, we want to minimize the cost of the more expensive edge.

5.2 Formulation II with 3-index Variables

In this section we propose a formulation where we use three index variables. Before presenting the selected variables, we define:

$$O_u := \sum_{v \in V; v>u} r_{u,v} \qquad \forall u \in V, \tag{5.9}$$

which represents the amount of requirement that must go out from a node u.

We use the following decision variables:

- x_{ij}, a binary decision variable which indicates if the edge (i, j) is in the tree. It is defined for all $i, j \in V, i < j$.
- f_{uij}, amount of flow with origin in u that circulates through the directed arc (i, j). It is defined for all $u, i, j \in V, i \neq j$.
- y_{uij}, a binary decision variable which is equal to 1 if the directed arc (i, j) is used to send flow originated in u. It is defined for all $u, i, j \in V$.

Then, a valid formulation is:

Set of constraints and variable domains.

$$\sum_{j \in V \setminus \{u\}} f_{uuj} = O_u \qquad \forall u \in V \tag{5.10}$$

$$\sum_{j \in V \setminus \{i,u\}} f_{uji} - \sum_{j \in V \setminus \{i,u\}} f_{uij} = r_{ui} \qquad \forall u, i \in V \tag{5.11}$$

$$f_{uij} \leq M y_{uij} \qquad \forall u, i, j \in V, j \neq i \tag{5.12}$$

$$\sum_{i,j \in V; j \neq i, j \neq u} y_{uij} = n - 1 \qquad \forall u \in V \tag{5.13}$$

$$y_{uij} + y_{uji} \leq x_{ij} \qquad \forall u, i, j \in V, i < j \tag{5.14}$$

$$\sum_{i,j \in V; i<j} x_{ij} = n - 1 \tag{5.15}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V, i<j \qquad (5.16)$$

$$y_{uij} \in \{0,1\} \qquad \forall u,i,j \in V \qquad (5.17)$$

$$f_{uij} \geq 0 \qquad \forall u,i,j \in V \qquad (5.18)$$

For each $u \in V$, constraints (5.10) require that the total flow which leaves node u is equal to the value O_u, while (5.11) guarantee that the flow with origin at u and arrives to i is at least the requirement r_{ui}. We relate the f and the y variables with (5.12), by activating only the arcs used to send the flows. (5.13) are cardinality constraints and, together (5.10)-(5.12), guarantee that, for each $u \in V$, the flow O_u is sent through arcs that define a spanning tree. Constraints (5.14) connect x with y variables by forcing that if an arc is used to send some flow, then the associated edge is in the tree and impose that every edge can be used only in one direction. Constraint (5.15) ensures that the total number of the edges in the tree is $n-1$, namely the solution represents a spanning tree. Finally, we complete the model with variable domains (5.16)-(5.18).
In constraints (5.12) M is a sufficiently large constant and its value is very important because it can affect the effectiveness of the constraints in solution algorithms. Thus, we have chosen a value for M that represents the maximum value that the flow may take.

$$M = O_u.$$

There are $m + 2n^2(n-1) = \mathcal{O}(n^3)$ variables and $\mathcal{O}(n^3)$ constraints.

Objective function for ME-OCSTP.

$$\min \max_{i,j \in V: i \neq j} c_{ij} \sum_{u \in V} (f_{uij} + f_{uji}) \qquad (5.19)$$

Now, the total communication cost of an edge is its cost multiplying by the total flow circulating through it. We want to minimize the cost of the more expensive edge.

For using formulation II for **MP-OCSTP**, we have to define a new decision variable:

- d_{ij}, distance in the tree between i and j. It is defined for all $i,j \in V, i \neq j$.

Objective function for MP-OCSTP.

$$\min \max_{i,j \in V: i \neq j} r_{ij} d_{ij} \qquad (5.20)$$

Now, the cost for sending one unit of flow from i to j is represented by the variable d_{ij}. Thus, multiplying it by the requirement, we find the cost of the path. We want to minimize the cost of the more expensive path.

Moreover, we have to add to 5.10-5.18, the following **constraints**, which relate the new variables with the others:

$$d_{ij} \geq c_{ij} x_{ij} \qquad \forall i, j \in V, i < j \tag{5.21}$$

$$d_{ij} \geq d_{ik} + c_{kj} - M_2(1 - y_{ikj}) \qquad \forall i, j, k \in V, i \neq j \neq k \tag{5.22}$$

$$d_{ij} = d_{ji} \geq 0 \qquad \forall i, j \in V, i \neq j \tag{5.23}$$

Constraints (5.21) and (5.22) control the d variables. In particular, (5.21) guarantee that the distance between i and j is at least equal to the cost of the associated edge if it is activated, and (5.22) relate d and y variables imposing that if the arc (k, j) is used for sending some flow from i to j, then the distance between i and j is larger then the sum of the distance between i and k and the cost of the edge (k, j). Finally, we complete the model with variable d domains (5.23).
We observe that if the instance satisfy the triangular inequality, then the constraints (5.21) can be transform in

$$d_{ij} \geq c_{ij}. \tag{5.24}$$

In fact also if the edge (i, j) is not in the tree, surely the path that connects i and j is at least long/expensive as the direct link.
In constraints (5.22) M_2 is a sufficiently large constant and its value is very important. It must be such that the corresponding constraint becomes redundant when $y_{ikj} \neq 1$. The value we have use is:

$$M_2 = c_{kj} - \sum_{u \in V \setminus \{1\}} \max_{v : v \neq u} c_{uv}.$$

There are $m + 2n^2(n-1) + n(n-1) = \mathcal{O}(n^3)$ variables and $\mathcal{O}(n^3)$ constraints.

5.2.1 Valid Inequalities

In this section we propose some families of valid inequalities that can be used to reinforce formulation II.

(a) Vertex cutset inequalities. Each node $u \in V$ must have at least one arc that comes out from it, i.e.

$$\sum_{i \in V \setminus \{u\}} y_{uui} \geq 1. \tag{5.25}$$

Moreover, for all pairs $u, i \in V$ with $u \neq j$, exactly one arc emanating from u, must enter in i, i.e.

$$\sum_{j \in V \setminus \{i\}} y_{uji} = 1. \tag{5.26}$$

(b) Set cutset inequalities. As all nodes must be visited, for all $S \subset V$, there must be at least one arc that connects a node belonging to S with one in its complement,

$$x(\delta(S)) \geq 1, \tag{5.27}$$

which can be rewritten as

$$\sum_{i \in S, j \notin S : i<j} x_{ij} + \sum_{i \in S, j \notin S : i>j} x_{ji} \geq 1.$$

(c) Min-Cut inequalities Let be m_{ij} the value of the min-cut separating i and j in the original graph, relative to a capacity vector given by r. For all $i, j \in V, i < j$, the following inequalities are valid:

$$m_{ij} x_{ij} \leq \sum_{u \in V \setminus \{j\}} f_{uij} + \sum_{u \in V \setminus \{i\}} f_{uji} \tag{5.28}$$

$$\sum_{u \in V \setminus \{j\}} r_{uj} y_{uij} + \sum_{u \in V \setminus \{i\}} r_{ui} y_{uji} \leq \sum_{u \in V \setminus \{j\}} f_{uij} + \sum_{u \in V \setminus \{i\}} f_{uji} \tag{5.29}$$

Inequality (5.28) is non-redundant when the edge (i, j) is active. In particular, if the edge is active, it imposes that the total flow through (i, j) in both the directions, must be at least equal to the value of the min-cut which separates i and j.

On the other hand, with (5.29), we impose that the same flow is also at least equal to the sum of the all requests involving i or j, if the arc (i, j) is used for sending the flow originated at u.

(d) Minimum flows trough arcs. For all $u, i, j \in V, i < j$, if the edge (i, j) is used for sending flow originated in u, then the associated flow must be greater than or equal to the request between u and i and the request between u and j, i.e.

$$r_{uj} y_{uij} + r_{ui} y_{uji} \leq f_{uij} + f_{uji}. \tag{5.30}$$

Let $k_1 = \max\{m_{uj}, m_{ij}\}$ and $k_2 = \max\{m_{iu}, m_{ij}\}$. The following inequality is true:

$$k_1 y_{uij} + k_2 y_{uji} \leq \sum_{v \in V \setminus \{j\}} f_{vij} + \sum_{v \in V \setminus \{i\}} f_{vji}. \tag{5.31}$$

5.3 Formulation III with 2-index Variables

For this formulation, we assume that the solution is a directed tree rooted at a selected vertex. We choose vertex 1, without loss of generality.

We use the following variables:

- x_{ij}, a binary decision variable which indicate if the (directed) arc (i, j) is in the tree. It is defined for all $i, j \in V, i \neq j$.
- p_{ij}, a binary decision variable which is equal to 1 if the tree contains a (directed) path from i to j. It is defined for all $i, j \in V, i \neq j$. In particular, when p_{ij} is equal to one, we can say that i precedes j in the tree.
- d_{ij}, distance in the tree between i and j. It is defined for all $i, j \in V, i \neq j$.

Then, a valid formulation is:

Set of constraints and variable domains.

$$\sum_{i \in V \setminus \{j\}} x_{ij} = 1 \qquad \forall j \in V \setminus \{1\} \qquad (5.32)$$

$$x_{ij} \leq p_{ij} \qquad \forall i, j \in V, i \neq j \qquad (5.33)$$

$$p_{ij} + p_{ji} \leq 1 \qquad \forall i, j \in V, i < j \qquad (5.34)$$

$$p_{ij} + p_{jk} + x_{kj} \leq 1 + p_{ik} \qquad \forall i, j, k \in V, i \neq j \neq k \qquad (5.35)$$

$$d_{ij} \geq c_{ij} x_{ij} \qquad \forall i, j \in V, i \neq j \qquad (5.36)$$

$$d_{ij} \geq d_{ik} + c_{kj} - M(1 - x_{kj} + p_{ji}) \qquad \forall i, j, k \in V, i \neq j \neq k \qquad (5.37)$$

$$\sum_{i,j \in V; i<j} x_{ij} = n - 1 \qquad (5.38)$$

$$x_{i1} = 0 \qquad \forall i \in V, i \neq 1 \qquad (5.39)$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in V, i \neq j \qquad (5.40)$$

$$p_{ij} \geq 0 \qquad \forall i, j \in V, i \neq j \qquad (5.41)$$

$$d_{ij} = d_{ji} \geq 0 \qquad \forall i, j \in V, i \neq j \qquad (5.42)$$

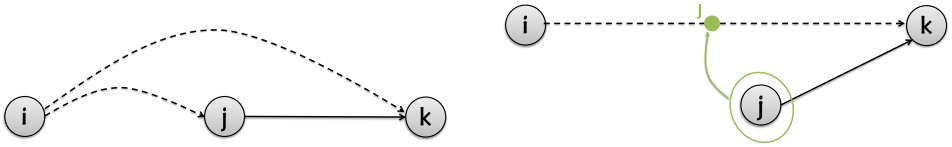
Constraints (5.32) guarantee that each node, except the root, has only and only one arc which enters in it. We relate the x and the p variables with (5.33). In fact if arc (i, j) is used, then there is a path that connects i with j. Constraints (5.34) ensure, for each pair of nodes (i, j), that either there is a directed path connecting j and i or i and j are not linked. Constraints (5.35)

ensure that the following two relations hold:

$$p_{ij} + x_{jk} \leq 1 + p_{ik} \qquad \forall i, j, k \in V, i \neq j \neq k \qquad (5.43)$$

$$p_{ik} + x_{jk} \leq 1 + p_{ij} \qquad \forall i, j, k \in V, i \neq j \neq k \qquad (5.44)$$

The first one ensures what we show in Figure 5.1a: if there is a directed path connecting i with j and the arc (j, k) is active, then a path from i to k must exist. Figure 5.1b, on other hand, shows that if the arc (j, k) is active and there is a path from i to k, then the node j is located in the path.



(a) *Explanation of* (5.43). (b) *Explanation of* (5.44).

Figure 5.1: Explanation of relation (5.43)-(5.44)

Variable d is controlled by constraints (5.36) and (5.37). In particular, (5.36) ensure that the distance between i and j is at least equal to the cost of the associated edge, when it is active; and (5.37) relate d to x and p by imposing that if j does not precede i and arc (j, k) is active, the distance between i and j is equal to the distance between i and k plus the cost of the edge (k, j), as shown in Figure 5.2. In all other cases this constraint is redundant. In (5.38)

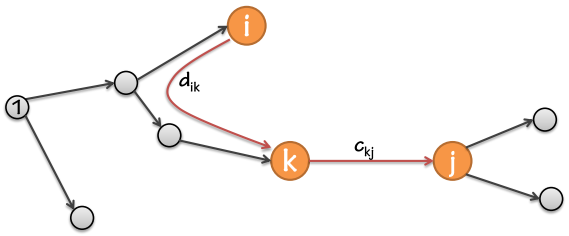


Figure 5.2: Meaning of constraints (5.37)

and (5.39) we impose that the solution is a spanning tree rooted at vertex 1. Finally, we complete the model with spanning tree constraint and variable domains (5.40)-(5.42).
We observe that if the instance satisfy the triangular inequality, then the constraints (5.21) can be transform in

$$d_{ij} \geq c_{ij}. \qquad (5.45)$$

In fact also if the edge (i, j) is not in the tree, surely the path that connects i and j is at least long/expensive as the direct link.
In constraints (5.37) M is a sufficiently large constant. The value we have chosen is:

$$M = c_{kj} - \sum_{u \in V \setminus \{1\}} \max_{v: v \neq u} c_{uv}$$

In this model, there are $3n(n-1) = \mathcal{O}(n^2)$ variables and $1 + 2m + n(n-1) + 2n(n-1)(n-2) + 2(n-1) = \mathcal{O}(n^3)$ constraints.

Objective function.

$$\min \max_{i,j \in V: i \neq j} r_{ij} d_{ij} \tag{5.46}$$

The variable d_{ij} represent the cost for sending one unit of flow from i to j. Thus, multiplying it by the requirement, we find the cost of the path. We want to minimize the cost of the more expensive path.

For using formulation III for **ME-OCSTP**, we have to define a new decision variable:

- f_{ij}, amount of flow that circulates between i and j. It is defined for all $i, j \in V$.

Objective function for ME-OCSTP.

$$\min \max_{i,j \in V: i \neq j} c_{ij}(f_{ij} + f_{ji}) \tag{5.47}$$

The total communication cost of an edge is its cost multiplying by the total flow circulating through it. We want to minimize the cost of the more expensive edge.

Moreover, we have to add to 5.32-5.42, the following **constraints**:

$$f_{ij} \geq R(S) \left(x_{ij} - \sum_{\substack{u \in S, v \in S^c, \\ (u,v) \neq (j,i)}} x_{uv} + x_{vu} \right) \qquad \begin{matrix} \forall S \subset V, \forall i,j \in V: \\ 1, i \in S^c, j \in S \end{matrix} \tag{5.48}$$

$$f_{ij} \geq 0 \qquad \forall i, j \in V \tag{5.49}$$

Constraints (5.48) guarantee that if arc (i, j) is active and it is the unique arc that connects S^c with S, the flow circulating through it is at least equal to $R(S)$. $R(S)$ is the total requirements that have to go from a node in S^c to a

node in S and it is:

$$R(S) = \sum_{u \in S^c, v \in S} (r_{uv} + r_{vu})$$

Finally, we complete the model with variable d domains (5.49). We observe that the number of constraints (5.48) is exponential in n.
There are $4n(n-1) = \mathcal{O}(n^2)$ variables and an exponential number of constraints.

5.3.1 Valid Inequalities

In this section we propose some families of valid inequalities that can be used to reinforce formulation III.

(A) The first one is a reinforcement of constraints (5.36) since this includes it. For all $i, j \in V, i < j$, we can impose that if there is not a direct connection between i and j, they are connected through at least one other node. Then, their distance is at least the minimum sum of costs between two arcs which have i or j as extremity.

$$d_{ij} \geq c_{ij}(x_{ij} + x_{ji}) + M^2_{ij}(1 - x_{ij} - x_{ji}), \tag{5.50}$$

where $M^2_{ij} = \min_{k \neq i,j}\{c_{ik} + c_{kj}\}$.

(B) All nodes must have as predecessor the root, thus it holds that:

$$d_{1j} \geq \sum_{i \neq j} Cm_i p_{ij} + \sum_{i \neq j} (c_{ij} - Cm_i)x_{ij}, \tag{5.51}$$

where $Cm_i = \min_{k \neq i} c_{ik}$. If we consider a node j, directly connected to 1, their distance must be at least equal to minimum cost of the arcs outgoing from 1. Instead, if j is not a node directly connected to 1, its distance from 1 must be at least equal to the sum of the minimum cost arc incident to the nodes in the path from 1 to j plus the cost of the unique arc incoming j.

(C) For the instances that satisfy the triangular inequality, for every 3-vertex set $\{i, j, k\}$

$$\sum_{\{u,v\} \subset \{i,j,k\}} r_{uv} d_{uv} \geq T^3_{ijk}, \tag{5.52}$$

where $T^3_{ijk} = \min\{\sum_{u,v \in T} r_{uv} c_{uv} + \sum_{u,v \notin T} r_{ij} M^2_{ij} | \text{T spanning tree on} \{i, j, k\}\}$.

(D) Cut-set inequalities. Since all nodes must be visited, for all $S \subset V \backslash \{1\}$, there must be at least one arc that connects a node not belonging to S with one that belongs to S,

$$x(\delta^-(S)) = \sum_{i \notin S, j \in S} x_{ij} \geq 1. \tag{5.53}$$

(E) Minimum distance inequalities. This inequality is valid only for instances that satisfy the triangular inequality. To use it we have to resolve iteratively the relaxation of the problem. Called $G^* = (V, E^*)$ with $E^* \subset E$, the solution tree of an iteration we can add:

$$d_{ij} \geq c_{ij} + \left(d_{ij}^{G^*} - c_{ij}\right) \left(\sum_{e \in E^*} x_e - (n-2)\right). \tag{5.54}$$

5.4 Addition of Valid Inequalities

The families (b) and (D) have exponential size and thus they have to be separated (see Section 3.2.3). As explained in 3.2.1 this can be done by finding the Gomory-Hu tree and then by identifying the minimum cut separating each pair of nodes.
In our problems, first in the initial formulation we add the subset of these inequalities associated with singletons, $S = \{i\}$ for all $i \in V$. Let (x_0, y_0, T_0) be the current solution with an objective value $Z(T_0)$.
Then, we control if T_0 violates some of inequalities building for the current solution the Gomory-Hu Tree relative to a capacity vector given by x_0. Then, we find the Gomory-Hu Tree relative to the capacity vector x_0. If there is a min-cut (S, S^c) with value smaller than 1, then the inequality of this type associated with S is violated by (x_0, y_0, T_0).
The algorithm repeats this procedure until it finds no violated constraints. If the last solution T_l is generated by a vector with x_l and y_l integer, then $Z(T_l)$ is the optimal solution. Otherwise, $Z(T_l)$ is only a lower bound of the optimal solution.

Instead, to add the family (E), we have to solve iteratively the relaxation of the problem. At each iteration we find a current solution $G_k = (V, E_k)$, which is generated by the vectors x_k, p_k and d_k. Since this is a solution of the relaxed problem, d_k is not really the vector with the distances between each pair of nodes in G_k. Thus, we have to calculate the really distance in G_k of each pair of nodes.

Moreover in the case of formulation III for ME-OCSTP, the family of constraints 5.48 has an exponential number of inequalities. For this reason, we cannot solve the instances with all the constraints. Hence, our goal is to find a relatively small subset of these constraints that determine the optimal solution. In particular, we solve it iteratively. In this case we are not relaxing the integrality constraints; at each iteration we are solving an integer linear program.
First, we only include those inequalities associated with sets S such that:

- $|S^c| = 1$;
- $|S^c| = 2$;
- $|S| = 1$.

Note that we are adding only $(n-1) + 2(n-1)(n-2) + (n-1)^2 \approx \mathcal{O}(n^2)$ constraints.
Moreover we add two families of constraints that are true:

$$f_{i1} = 0 \qquad \forall i \in V \tag{5.55}$$

$$f_{ij} \geq \sum_{u \in V} (r_{uj} + r_{ju}) p_{uj} + \sum_{\substack{v \in V: \\ v \neq j}} (r_{iv} + r_{vi}) p_{iv} - MF(1 - x_{ij}) \qquad \substack{\forall i,j \in V: \\ i \neq j; i \neq 1} \tag{5.56}$$

Since the vertex one is the root of the tree, (5.55) guarantee that no flow enters in in the root. If the arc (i, j) is active, the constraint (5.56) imposes that a minimum amount of flow circulates through it. When (i, j) is not active, the constraint is redundant and for this we have chosen a value for MF such that:

$$MF_{ij} = \sum_{u \in V: u \neq i} (r_{ui} + r_{iu}) + \sum_{v \in V: v \neq i, v \neq j} (r_{vj} + r_{jv}).$$

Thus, we are adding other $(n-1) + n(n-1) = \mathcal{O}(n^2)$ constraints.

The general idea of the algorithm used for adding only a "small" number of these constraints is the same presented in Section 3.2.3. Note that in this case if there are constraints violated, these are added to the current system. Otherwise the current solution is optimal. Since with some instances the computational time is long, we have chosen that after an hour the system returns the current solution. In this case the current solution represents a lower bound of the optimal solution.
We observe that we find the violated constraints only for the edges in the current solution because for the edges which are not in the solution these constraints are redundant.

Chapter 6

Heuristics

In this chapter we present some heuristic algorithms which can be used to find upper bounds of the optimal value of MP-OCSTP and ME-OCSTP.
In Section 6.1, we explain what a heuristic algorithm is, then in Section 6.2 and 6.3 we illustrate all the algorithms we have implemented.

6.1 Heuristic Algorithm

A heuristic algorithm is any method that produces a feasible solution of a given problem. Generally, a heuristic algorithm solves an instance in polynomial time on the size of the instance, but in complex problems this constraint can be relaxed with the request that the algorithm is "fast".
When a problem is $\mathcal{NP}$-hard, there is not the certainty that the algorithm can determinate a feasible solution.
In our cases, a feasible solution is represented by any spanning tree; since we have a complete graph, it is always possible to find a spanning tree.
In general, given an instance I of a minimization problem (as the MP-OCSTP and ME-OCSTP are) a heuristic algorithm H produces a solution with a value $z^H(I)$ such that:

$$z^H(I) \geq z^*(I), \tag{6.1}$$

where $z^*(I)$ is the optimal value of I. Thus, the algorithm determines an upper bound of the optimal value.
Broadly speaking, there exist two types of heuristic algorithms:

1. Greedy: a type of algorithm that produces a solution through a sequence of partial decisions (locally optimal), without going back to modify the

made decisions.

2. Local-search: it examines a feasible solution and iteratively tries to improve it by making simple moves.

6.2 Greedy Algorithms

In this section we propose some greedy algorithms for the problem that we study.
Since each algorithm determines a spanning tree, it is a feasible solution for both problems, MP-OCSTP and ME-OCSTP. Obviously, if an algorithm is conceived for one specific problem, it may produce an upper bound of poor quality for the other problem.

6.2.1 Minimum Spanning Tree

Since in [14] the author suggested that some heuristics for the OCSTP could improve by starting with an Minimum Spanning Tree (MST), the first greedy algorithms that we propose are based on this observation. Thus, we present a simple description of the MST.
In the MST, there is a graph $G' = (V', E')$ and each edge has an associated weight w_e. The algorithm we are presenting builds a solution iteratively starting from the empty one and, at each iteration, adds a new edge to the current partial solution. To guarantee the algorithm correctness two criteria must be established:

i) The order according to which edge should be considered to be added to the partial solution.

ii) How to decide if an edge can be added to the current partial solution.

With respect to the order, we select the edge with the smallest weight among the edges not yet chosen. An edge is added to the current solution only if the new set of edges can be part of a feasible solution, namely the edge added to the solution must not form any cycle with those edges already chosen. Thus, this algorithm, which is the well-known Kruskal's algorithm ([13]), is the following:

Algorithm 6.1: Kruskal's Algorithm

```
1  input: G'=(V', E')
2  begin
3      T:= ∅ , S:=E
4      while  T ≠ ∅
```

```
5         e∈argmin(w_e, e ∈ S);
6         S:=S\{e};
7         if T ∪{e} does not contain cycles then T:= T∪{e}
8       end
9       return T;
10    end
```

To find the MST, the well-known Prim's algorithm is more efficient than the Kruskal's algorithm; but, because the instances that we use have a small size, what algorithm we use is irrelevant.

Now, we propose some heuristic algorithms based on the idea of finding an MST relative some modified cost function.

Heuristic I

In the first heuristic that we propose, the spanning tree is built with Kruskal's algorithm using as weights the original costs. So, we refer to **(HE-I)** to indicate:

(HE-I): *The heuristic where edges are considered by increasing values of their costs, and an edge is added only if it does not generate a cycle with other edges already chosen.*

The complexity of (HE-I) is $\mathcal{O}(mlog(n) + m^2log(n))$.

Heuristic II

The rationale for this heuristic is that in the MP-OCSTP if a pair has a big communication requirement, the nodes of this pair should be directly connected. In fact, in the MP-OCSTP if the path that connects two nodes with a big requirement is long (thus, more expensive), it is probably that the value of the most expensive path of the tree increases. For this reason, we propose the following heuristic, which is a minor variation of the Kruskal's algorithm:

(HE-II): *In this heuristic, the weights of the graph are given by the communication requirement. So, we consider the edges ordered from the greatest requirement to the smallest one, and an edge is added only if it does not generate a cycle with other edges already chosen.*

The complexity of (HE-II) is $\mathcal{O}(mlog(n) + m^2log(n))$.

Heuristic III

The following heuristic has been conceived for the MP-OCSTP. First of all, we assume that the cost function satisfies the triangle inequality:

$$c_{ij} \leq c_{ik} + c_{kj} \qquad \forall i, j, k \in V \tag{6.2}$$

Therefore if in the tree T two nodes i and j are not directly connected, the communication cost over T of (i, j) is greater than $c_{ij} \cdot r_{ij}$. Consequently, a lower bound on the optimal value is:

$$lb := \max_{i,j \in V : i<j} c_{ij} r_{ij} \tag{6.3}$$

Let $(\bar{i}, \bar{j})$ denote the pair of nodes that produces the maximum in (6.3). Consequently, if we do not connect directly $\bar{i}$ and $\bar{j}$, the optimal value will be greater than lb.
For this reason, we propose a heuristic based on Kruskal's algorithm, where the weight of the edge (i, j) is equal to $c_{ij} \cdot r_{ij}$.

(HE-III): *In this heuristic, we consider the edges ordered from the greatest weight to the smallest one, and an edge is added only if it does not generate a cycle with other edges already chosen.*

The complexity of (HE-III) is $\mathcal{O}(mlog(n) + m^2 log(n))$.

6.2.2 Heuristic with Costs Update

Now, we propose another heuristic (**HE-IV**) for the MP-OCSTP.

This heuristic is based on the following idea.

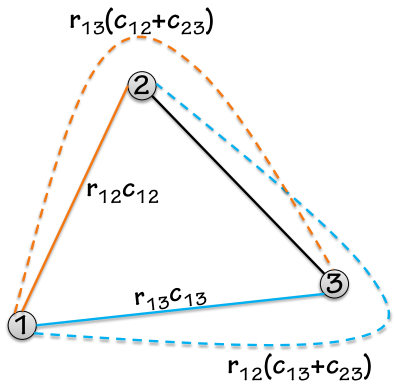


Figure 6.1: Main idea of **HE-IV**.

Suppose $n = 3$ and we have already decided that the first edge that takes part of the tree is that which gives lb in (6.3). Then we have to choose what edge add. This situation is shown in Figure 6.1. Here we assume that the initial edge is $(2, 3)$.
If we decide to add the edge $(1, 3)$, then implicitly the communication cost over T of the pair $(1, 2)$ is established. The same holds true for the edge $(1, 2)$.

We want to add the edge that increases less the current value of the objective function taking into account the above comment. First, we create an auxiliary matrix W:

$$W = R \cdot C = \begin{pmatrix} 0 & r_{12}c_{12} & r_{13}c_{13} & \dots & r_{1n}c_{1n} \\ 0 & 0 & r_{23}c_{23} & \dots & r_{2n}c_{2n} \\ 0 & 0 & 0 & \dots & r_{3n}c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

Then, we choose the edge with maximum $r_{ij}c_{ij}$:

$$W = \begin{pmatrix} 0 & r_{12}c_{12} & \boxed{r_{13}c_{13}} & \dots & r_{1n}c_{1n} \\ 0 & 0 & r_{23}c_{23} & \dots & r_{2n}c_{2n} \\ 0 & 0 & 0 & \dots & r_{3n}c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

Now, we update M taking into account what we have said before. Thus, we change an entry value in the matrix if the related edge is connected with the edge already chosen and the path, which is generated by adding it, is greater than the current value.

$$W = \begin{pmatrix} 0 & \begin{matrix} max(r_{12}c_{12}, \\ r_{23}(c_{12}+c_{13})) \end{matrix} & r_{13}c_{13} & \dots & \begin{matrix} max(r_{1n}c_{1n}, \\ r_{n3}(c_{1n}+c_{13})) \end{matrix} \\ 0 & 0 & \begin{matrix} max(r_{23}c_{23}, \\ r_{12}(c_{23}+c_{13})) \end{matrix} & \dots & r_{2n}c_{2n} \\ 0 & 0 & 0 & \dots & r_{3n}c_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}.$$

In the solution we insert the edge, connected to the edge already chosen with the smallest value in the matrix.
Then, we repeat the matrix update and the choice of thr edge to add (obviously, it must not create a cycle) until the solution has $n - 1$ edges.

The complexity of (HE-IV) is $\mathcal{O}(n^2 + m^2 log(n))$.

6.2.3 Star-Tree Heuristic

In Chapter 4 we have said that there exist particular situations where the optimal solution for MP-OCSTP is a star-tree. For this reason, we propose a simple heuristic (**HE-V**) which produces a star-tree.
This algorithm builds all the star-trees of the graph changing the unique inner node of the tree and for each of them computes:

- For the MP-OCSTP, the cost of the most expensive path in the current tree.
- For the ME-OCSTP, the cost of the most expensive edge in the current tree.

The output is the star-tree where the cost of the most expensive path/edge is minimum. Moreover, the complexity of (HE-V) is $\mathcal{O}(n^3)$.

6.2.4 Heuristics based on Gomory-Hu Tree

Now, we consider the ME-OCSTP. We observe that if we choose to put in the tree the edge (i, j), the amount of flow circulating through it could be at least the capacity b of the minimum cut which separates i and j with respect to the requirements.
Based on this observation, we propose a heuristic based on Kruskal's algorithm 6.1, where the weight of the edge (i, j) is equal to $c_{ij} \cdot b_{ij}$.

(HE-VI): *In this heuristic, we consider the edges ordered from the smallest weight to the greatest one, and an edge is added only if it does not generate a loop with other edges already chosen.*

As we said in 3.2.1, to compute the capacity of the minimum cut for all pairs of nodes of the graph, it is better to find out the Gomory-Hu Tree. Therefore, the complexity of (HE-VI) is $\mathcal{O}(nmC + n^2 + m^2 log(n))$.

Moreover, we observe that the Gomory-Hu tree is a spanning tree. Thus, it defines a feasible solution of MP-OCSTP and ME-OCSTP.

(HE-VII): *In this heuristic, we find out the Gomory-Hu Tree and compute its objective function value.*

The complexity of (HE-VII) is $\mathcal{O}(nmC + m^2 log(n))$.

6.3 Local-Search

In this section, we present two local-search algorithms, one for MP-OCSTP and the second one for ME-OCSTP. Both algorithms perform edge interchanges, in which an edge in the current solution is substituted by an edge outside it.

6.3.1 Local-Search for MP-OCSTP

Given a feasible solution, this algorithm tries to improve it iteratively. Consider the feasible solution in Figure 6.2, where the red path is the most expensive. Let $z(T_0)$ denote its cost.

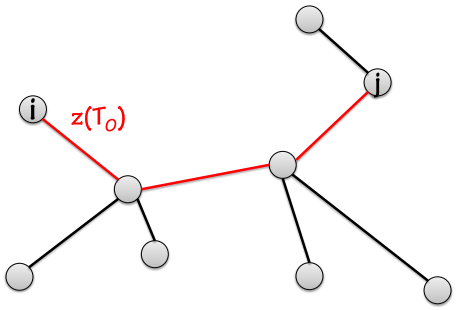


Figure 6.2: Example of feasible solution for MP-OCSTP

We want to modify the solution so that:

- The cost of new path connecting i with j is reduced;
- The cost of the new most expensive path is smaller than $z(T_0)$.

At each iteration we try to put in the tree the direct edge (i, j), as shown in Figure 6.3. If G satisfies the triangle inequality, we know that the first condition above is satisfied.

Now, we have to delete one of edges in the cycle that has appeared.

The algorithm tries to delete the edges of the cycle, at a time. If, deleting an edge, reduces the cost of the new most expensive path $z(T_1)$, i.e.

$$z(T_1) \leq z(T_0),$$

then the new feasible solution is T_1 and the algorithm restarts. Otherwise, the algorithm tries to delete another edge of the cycle.
When no other edge can be deleted, the algorithm returns the value $z(T_k)$,

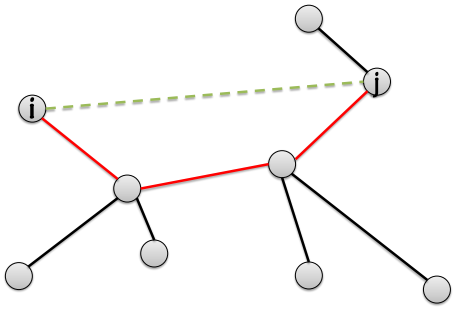
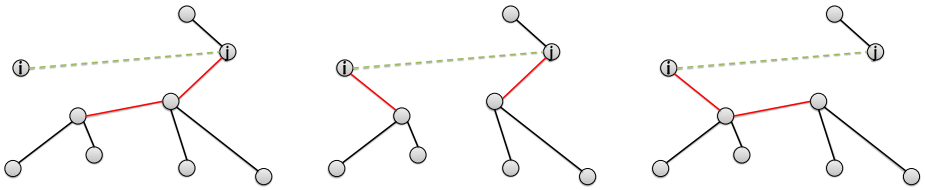


Figure 6.3: Example of the edge that the Local-Search algorithm for MP-OCSTP tries to add.



(a) . (b) . (c) .

Figure 6.4: Example of the edges that the Local-Search algorithm for MP-OCSTP tries to delete.

namely the cost of the most expensive path of the solution T_k obtained at k^{th} iteration.

6.3.2 Local-Search for ME-OCSTP

Given a feasible solution, this algorithm tries to improve it iteratively.

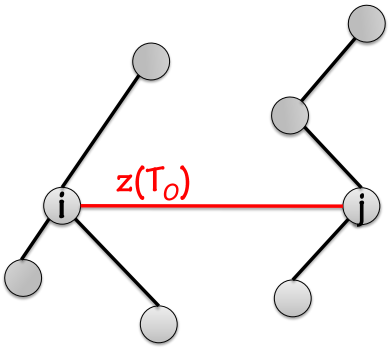


Figure 6.5: Example of feasible solution for ME-OCSTP

Consider the feasible solution in Figure 6.5, where the red edge is the most expensive. Let $z(T_0)$ denote its cost. We observe that $z(T_0)$ is equal to the cost c_{ij} multiplied by the sum of all requirements between a node of the subset S and a node in its complement S^c. Thus, to improve $z(T_0)$, we try to substitute the edge (i, j) with another edge of the cut (S, S^c) as shown in Figure 6.6.

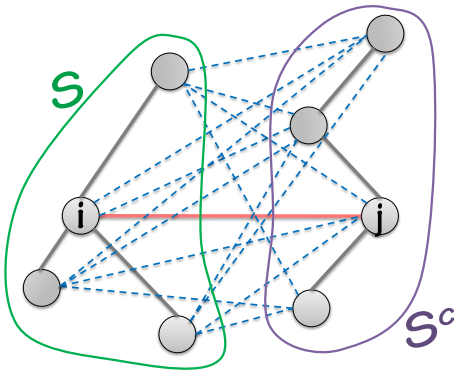


Figure 6.6: Example of the edges in the cut which separates i and j.

Since the amount of flow that would circulate through the new edge is the same as that circulating in T through (i, j), it can be substitute only with an edge with a smaller cost (Figure 6.7).

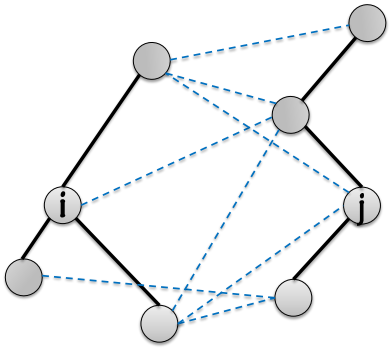


Figure 6.7: Example of the edges that the Local-Search algorithm for MP-OCSTP tries to add.

The algorithm tries to insert one edge at a time. If, inserting an edge, the cost of the new most expensive edge $z(T_1)$ is such that

$$z(T_1) \leq z(T_0),$$

then the new feasible solution is T_1 and the algorithm restarts. Otherwise the algorithm tries to insert another edge.

When no other edge can be inserted, the algorithm returns the value $z(T_k)$, namely the cost of the most expensive edge of the solution T_k obtained at k^{th} iteration.

Chapter 7

Implementation and Results

In this chapter, we will describe the computational experiments that have been run, and will summarize the obtained numerical results.
Firstly, in Section 7.1 we describe the instances which we have used to solve the MP-OCSTP and the ME-OCSTP and in Section 7.2 introduce the programs used to implement the formulations and the heuristic algorithms.
Then, in Sections 7.3-7.5 we explain how we have implemented the three formulations and the results they produce and in Section 7.6 we compare them.
Finally, in Sections 7.7-7.8 we present the results obtained applying all the Heuristic Algorithms to solve these problems.

7.1 Analysis of existing Problems Instances

Several sets of instances for the OCSTP have been proposed in literature by various authors (benchmark instances). We have used some of these and new ones to analyze the results of the three formulations, which have presented in Chapter 5, and of the heuristics proposed in Chapter 6. To the best of our knowledge, the optimal solutions to the MP-OCSTP and ME-OCSTP are not known. Since the computational time to solve instances with more of 16 nodes is great, the results which we present in this Chapter are obtained using small instances.
The first set of instances we have used was described by Palmer. In his thesis he described instances with 6 (PALM06), 12 (PALM12), 24, 47 and 98 nodes. The communication requirements are inversely proportional to the distances between the nodes. The nodes correspond to cities in the United States and the distance between the nodes are obtained from a tariff database. We have

used only PALM06 and PALM12, but we have created also PALM09, PALM10 and PALM16 deleting nodes from instances of larger sizes.
In 2001, Raidl proposed several test instances. The costs and the communication requirements were generated randomly. They are uniformly distributed in the interval $[0, 100]$. Moreover, we have generated the following sets of benchmark instances:

- EUCLUN06, EUCLUN09, EUCLUN12 and EUCLUN15: in these instances the x and y coordinates are generated randomly in the xy plane. The cost matrix respects the distances of the nodes in the plane.
- EUCLINVPROP06, EUCLINVPROP09, EUCLINVPROP12 and EUCLINVPROP15 are instances where the x and y coordinates of nodes are generated randomly in the xy plane. The cost matrix respects the distances of the nodes in the plane. Moreover the communication requirements are inversely proportional to the costs.

In all our experiments we have set the maximum computing time to 60 minutes.

7.2 Implementation

The formulations and the families of valid inequalities presented in the previous sections have been solved using IBM ILOG CPLEX OPTIMIZATION STUDIO 12.5. It consists of the CPLEX OPTIMIZER for mathematical programming, the IBM ILOG CPLEX CP OPTIMIZER for constraint programming, the OPTIMIZATION PROGRAMMING LANGUAGE (OPL), and a tightly integrated IDE.
The heuristic algorithms have been implemented using MICROSOFT VISUAL STUDIO 2013. It supports different programming languages, but we have implemented all the algorithms in C++.
All experiments have been run on an Intel Core i7-3517 2.4 GHz with 4 GB Ram and operating system Windows 8.1 Pro, 64 bit.

7.3 Formulation I

In this section we present how the Formulation I has been implemented and the results we have obtained using it and its linear programming (LP) relaxation.

7.3.1 Implementation

CPLEX Optimization Studio is organized in *projects*. Each project contains the files `.dat`, where there are the data of the current instance, and a file `.mod` with the code of the formulation.
Here, we present the code for this formulation. First we declare the data of the problem that the program reads in the file `.dat`.

```
//DATA
int N= ...;
range Nodes= 1..N;
float c[Nodes, Nodes]= ...;
float r[Nodes, Nodes]= ...;
```

Since we have to work with Edges and Arcs, we define the appropriate structures for them and the decision variables of this formulation.

```
tuple edge  {int ini; int fin;}
{edge} Edge={<ini, fin> | ini, fin in Nodes: ini<fin};
{edge} Arcs={<ini, fin> | ini, fin in Nodes: ini!=fin};

//VARIABLES
 dvar boolean X[Edge];
 dvar boolean Y[Edge][Arcs]; //Edge: path, Arcs: edge
 dvar float+ z;
```

Now, we have to declare the objective function of the problem.
Since MP-OCSTP and ME-OCSTP have a MIN-MAX objective function, we have defined a variable z, which will be minimized. Then, among the constraints for the MP-OCSTP we have to impose that z is greater than the cost of each path in the solution, while for the ME-OCSTP z has to be greater than the cost of each edge in the solution.

```
//OBJECTIVE
minimize z;

//CONSTRAINTS
subject to{

  forall(e in Edge:r[e.ini, e.fin]>0) //for MP-OCSTP
```

```
22      z >= (r[e.ini, e.fin]* sum(e1 in Arcs) c[e1.ini, e1.
            fin]*Y[e][a1]);

24    // forall(a in Edge) //for ME-OCSTP
        // z>= c[e.ini, e.fin]* sum(e1 in Edge) (r[e1.ini, e1.
            fin]*(Y[e1][e]+Y[e1][<e.fin,e.ini>]));
26
    forall(e in Edge: r[e.ini, e.fin]>0)
28    sum(j in Nodes: j!=e.ini) Y[e][<e.ini, j>] ==1;

30  forall(e in Edge: r[e.ini, e.fin]>0, j in Nodes: j!=e.
        ini && j!=e.fin)
      sum(i in Nodes: i!=j && i!=e.fin) Y[e][<i,j>] - sum(k
          in Nodes: k!=j && k!=e.ini) Y[a][<j,k>] ==0;
32
    forall(e, e1 in Edge)
34    Y[e][e1]+ Y[e][<e1.fin, e1.ini>] <= X[e1];

36  sum(e in Edge) X[e]==N-1;
    }
```

7.3.2 Computational Results

Now, we show the results that we have obtained with the code presented above. Moreover we have solved the linear relaxation presented in 3.2.2 of this formulation.

In Tables 7.1-7.2, we present the results obtained solving both the problems and their relaxations. The meaning of the columns in the tables is the following. Column *Optimal* contains the optimal values of the instances; Column *Time (sec)*, the computing times needed to obtain the optimal values (in seconds), and column *Relaxation* the objective function values of the LP relaxation. The percentage of the LP values with respect to the optimal values are given in column *%*.

First, we note that the time to solve the instances increases very rapidly with the size of the instances. For the MP-OCSTP the computing times are still acceptable while for the ME-OCSTP they increase too faster and for this reason it is impossible to solve instances with more than 10 nodes in reasonable times. The times needed to solve the LP relaxations are negligible as compared to the

Table 7.1: Results of formulation I for MP-OCSTP.

Formulation I: MP-OCSTP				
Instance	**Optimal**	**Time** (sec)	**Relaxation**	**%**
PALM06	133300	0.765	133300.00	100.00%
PALM09	141900	9.828	141900.00	100.00%
PALM10	141900	54.297	141900.00	100.00%
PALM12	141900	951.515	141900.00	100.00%
PALM16	106800	3600	106800.00	100.00%
EUCLINVPROP06	21312	1.063	17078.98	80.14%
EUCLINVPROP09	24992	16.953	18307.39	73.25%
EUCLINVPROP12	28458	763.734	19086.47	67.07%
EUCLINVPROP15	27248	3600	19224.68	70.55%
EUCLUN06	622500	1.234	560101.00	89.98%
EUCLUN09	994788	221.609	987540.00	99.27%
EUCLUN12	994788	3600.000	987540.00	99.27%
EUCLUN15	1091200	3600	1018160.00	93.31%
RAIDL06	530483	0.484	530483.00	100.00%
RAIDL09	598186	60.578	530483.00	88.68%
RAIDL12	451005	123.532	387364.00	85.89%
RAIDL15	381220	3600	282240.00	74.04%

overall times, so they are not shown.

In Table 7.1 we see that for the MP-OCSTP the the LP relaxation produces the optimal value with the Palmer series. With the other instances (EUCLUN and RAIDL) the percentages decrease, but they are still good ($85-90\%$). Unfortunately the results obtained with EUCINVPROP are worst and the percentages are around the 70%, on average.

For the ME-OCSTP (Table 7.2) since the time to solve a medium size instances is high, we do not have all the optimal values with which to compare the results of the relaxation. For this reason, when we do not know the optimal value,

we put in the Table the value of the best-known solution and we indicate them with an "*". Nevertheless, the percentages presented are very low and it indicates that this formulation does not work well if we want to minimize the cost of the most expensive edge.

Table 7.2: Results with formulation I for ME-OCSTP.

Formulation I: ME-OCSTP			
Instance	**Optimal/ Best-known***	**Relaxation**	**%**
palm06	180600	74200.51	41.09%
palm09	320342	84834.41	26.48%
palm10	326502	79388.71	24.31%
palm12	820301*	74504.93	9.08%
palm16	56700*	30175.74	53.22%
EuclInvProp06	75542	16426.95	21.75%
EuclInvProp09	134846	16659.01	12.35%
EuclInvProp12	272025*	17274.05	6.35%
EuclInvProp15	424402*	17750.09	4.18%
EuclUn06	1149358	327779.57	28.52%
EuclUn09	2382568	430522.39	18.07%
EuclUn12	3854732*	414776.20	10.76%
EuclUn15	13427700*	414550.62	3.09%
Raidl06	1572645	385427.39	24.51%
Raidl09	2117302*	330449.66	15.61%
Raidl12	4251531*	275727.84	6.49%
Raidl15	15799296*	239537.41	1.52%

7.4 Formulation II

We next present the results obtained with Formulation II and its relaxation. Moreover for this formulation, we have four families of valid inequalities (see Section 5.2.1) we can add to the LP relaxation to obtain a tighter lower bound.

The families (a), (c) and (d) can be added as a normal constraint, but family (b) is of exponential size and thus have to be added as explained in Section 5.4.

In the Tables 7.3-7.4, we present the results obtained for both problems and the values of their LP relaxations after adding the separated valid inequalities.

For the MP-OCSTP (Table 7.3), the LP relaxation applied to PALMER series reaches the optimal values; consequently it is unnecessary reinforce it with the valid inequalities.
This formulation returns different results depending on whether or not the instance satisfies the triangular inequality. In the first case, no inequality produces improvements. Therefore the LP bound of this formulation lies within $70 - 90\%$ of the optimal value (with the EUCINVPROOP the results are less good). We note that when we say that the valid inequality does not reinforce the lower bound, it does not mean that the spanning tree found will be the same, but only that the cost of the most expensive path does not change. When the instances do not satisfy the triangular inequality, even if the inequalities (a) and (b) reinforce the lower bound, no relaxation value overcomes 8% of the optimal value. Thus we can conclude that this formulation is not appropriate to solve randomly generated instances.
Analyzing Table 7.4 we note that the family of valid inequality (c) reinforces the lower bound also with the instances which satisfy the triangular inequality; but the few obtained percentages do not overcome 50% of the optimal value. On the contrary, there are instances where the percentages are smaller than 25%.

Table 7.3: Results with formulation II for MP-OCSTP.

Formulation II: MP-OCSTP												
	Optimal		**Relaxation**		**Relaxation+(a)**		**Relaxation+(b)**		**Relaxation+(c)**		**Relaxation+(d)**	
Instance	value	Time (s)	value	%	value	%	value	%	value	%	value	%
palm06	133300	1.297	133300	100.00%								
palm09	141900	25.938	141900	100.00%								
palm10	141900	92.844	141900	100.00%								
palm12	141900	2614.5	141900	100.00%								
palm16	106800	3600	106800	100.00%								
EuclInvProp06	21312	1.453	16328	76.61%	16328	76.61%	16328	76.61%	16328	76.61%	16328	76.61%
EuclInvProp09	24992	21.797	16328	65.33%	16328	65.33%	16328	65.33%	16328	65.33%	16328	65.33%
EuclInvProp12	28458	1080.469	18627	65.45%	18627	65.45%	18627	65.45%	18627	65.45%	18627	65.45%
EuclInvProp15	27248	3600	18627	68.36%	18627	68.36%	18627	68.36%	18627	68.36%	18627	68.36%
EuclUn06	622500	1.234	560101	89.98%	560101	89.98%	560101	89.98%	560101	89.98%	560101	89.98%
EuclUn09	994788	155.172	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%
EuclUn12	994788	2343.922	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%
EuclUn15	1091200	3600	1018160	93.31%	1018160	93.31%	1018160	93.31%	1018160	93.31%	1018160	93.31%
Raidl06	530483	0.937	24207.82	4.56%	39850.04	7.51%	39850.04	7.51%	24207.82	4.56%	24207.82	4.56%
Raidl09	598186	444.266	16850.91	2.82%	22112.36	3.70%	20920.68	3.50%	16850.91	2.82%	16850.91	2.82%
Raidl12	451005	1966.094	13003.65	2.88%	20411.29	4.53%	20411.29	4.53%	13003.65	2.88%	13003.65	2.88%
Raidl15	381220	3600	5344.49	1.40%	10851.92	2.85%	10851.92	2.85%	5344.49	1.40%	5344.49	1.40%

Table 7.4: Results with formulation II for ME-OCSTP.

Formulation II: ME-OCSTP											
	Optimal Best-known*	**Relaxation**		**Relaxation+(a)**		**Relaxation+(b)**		**Relaxation+(c)**		**Relaxation+(d)**	
Instance	value	value	%	value	%	value	%	value	%	value	%
palm06	180600	71575	39.63%	71575	39.63%	71575	39.63%	77205	42.75%	77278	42.79%
palm09	320342	83454	26.05%	83454	26.05%	83454	26.05%	83454	26.05%	85867	26.80%
palm10	326502	78963	24.18%	78963	24.18%	78963	24.18%	78963	24.18%	80555	24.67%
palm12	820301*	72323	8.82%	72323	8.82%	72323	8.82%	72323	8.82%	72908	8.89%
palm16	56700*	30176	53.22%	30176	53.22%	30176	53.22%	30176	53.22%	30176	53.22%
EuclInvProp06	75542	12323	16.31%	12323	16.31%	12323	16.31%	16100	21.31%	14212	18.81%
EuclInvProp09	134846	11852	8.79%	11852	8.79%	11852	8.79%	15137	11.23%	12376	9.18%
EuclInvProp12	272025*	11516	4.23%	11516	4.23%	11516	4.23%	15427	5.67%	12199	4.48%
EuclInvProp15	424402*	11249	2.65%	11249	2.65%	11249	2.65%	16535	3.90%	390804	92.08%
EuclUn06	1149358	327780	28.52%	327780	28.52%	327780	28.52%	327780	28.52%	332900	28.96%
EuclUn09	2382568	417455	17.52%	417455	17.52%	417455	17.52%	417455	17.52%	417455	17.52%
EuclUn12	3854732*	392512	10.18%	392512	10.18%	392512	10.18%	392512	10.18%	393595	10.21%
EuclUn15	13427700*	390804	2.91%	390804	2.91%	390804	2.91%	390804	2.91%	390804	2.91%
Raidl06	1572645	385427	24.51%	385427	24.51%	385427	24.51%	385427	24.51%	385427	24.51%
Raidl09	2117302*	330450	15.61%	330450	15.61%	330450	15.61%	330450	15.61%	330450	15.61%
Raidl12	4251531*	275546	6.48%	275546	6.48%	275546	6.48%	275546	6.48%	275546	6.48%
Raidl15	15799296*	227991	1.44%	227991	1.44%	227991	1.44%	227990.6044	1.44%	227991	1.44%

7.5 Formulation III

For this formulation, we have solved the MP-OCSTP and ME-OCSTP with different methods. For this reason, we present their implementation and results separately.

7.5.1 MP-OCSTP

Next, we present the results obtained with Formulation III and its LP relaxation for solving MP-OCSTP.
For this formulation, we have five families of valid inequalities (Section 5.3.1) we can add to the LP relaxation to obtain a stricter lower bound.
The families (A), (B) and (C) can be added as constraints in the initial formulation, whereas (D) and (E) can be added as presented in Section 5.4.
In particular, to add the family (E) at each iteration we have to calculate the really distance in G_k of each pair of nodes.

For this reason, we have applied the algorithm of Dijkstra for each node of G_k. This algorithm permits to calculate the distance *dist* between a node s and the others.

Algorithm 7.1: Dijkstra's Algorithm

```
1  input: G_k, dist, s
2  begin
3     Q=∅;
4     forall v∈V
5        dist[v]=+∞;
6     end
7     dist[s]=0;
8     Q=Q∪{s};
9     while Q≠∅
10       select u from Q such that d[i]=min{d[h]:h∈Q};
11       Q=Q\{u};
12       forall v∈δ(i)
13          if dist[v]>dist[u]+c[u,v]
14             then d[v]=d[u]+c[u,v];
15             if v∉ Q then Q=Q∪{V};
16          end
17       end
18    end
19    return dist;
20 end
```

The complexity of this algorithm depends on the data structures used to implement the set Q. Since, in IBM ILOG CPLEX Optimization Studio

12.5 is not possible create a priority queue, which reduces the complexity, our implementation of Dijkstra's Algorithm performs in $\mathcal{O}(n^2 + m)$ operations.

After computing the true distances d in G_k, we compare them with d_k and we add the inequalities for an edge (i, j) if $d_{k_{ij}}$ and d_{ij} differs at least ϵ. The value of ϵ depends on the instance. Since with some instances the inequality does not reinforce the lower bound, we have add all the inequalities for which the distances are different.
If there are no inequalities to add, the program returns the objective function value of current iteration. Otherwise the program solves again the instance with the inequalities added. In any case, after a prearranged number of iterations ($nmax$), the program returns the current solution. $nmax$ is the number of iterations that the system can solve in a hour.

In the Table 7.5, we present the results obtained solving the MP-OCSTP and its relaxation with valid inequalities.

With this formulation, the computation times to solve the instances increase with the size of the graph, but less rapidly than with the other formulations. Now, we analyze the results obtained relaxing the integrality constraints of the variables. The relaxation applied to the PALMER series is optimal (100%) and for this reason it is unnecessary to add the valid inequalities. With EUCINVPROP and EUCUN series no inequalities reinforce the lower bound obtained with the basic relaxation. In the case of inequalities (E) this is due to the fact that they increase the number of non-zero x variables, although they do not affect the objective function value for MP-OCSTP. Moreover the percentages of EUCINVPROP differ from those of EUCUN in around 10%.
With the RAIDL series we can not apply all the valid inequalities because this series does not satisfy the triangular inequality. In this case the family of inequalities (B) reinforce the lower bound, but it is not sufficient to conclude that this formulation works well with this type of instances. In fact the percentages do not overcome the 40%.

Table 7.5: Results obtained using the formulation III for MP-OCSTP.

Formulation III: MP-OCSTP														
	Optimal		**Relaxation**		**Relaxation+(A)**		**Relaxation+(B)**		**Relaxation+(C)**		**Relaxation+(D)**			
Instance	value	Time (s)	value	%	value	%	value	%	value	%	value	%		
palm06	133300	1.938	133300	100.00%										
palm09	141900	2.359	141900	100.00%										
palm10	141900	3.141	141900	100.00%										
palm12	141900	13.687	141900	100.00%										
palm16	106800	3600	106800	100.00%										
EuclInvProp06	21312	0.578	16328	76.61%	16328	76.61%	16328	76.61%	16328	76.61%	16328	76.61%	16328	76.61%
EuclInvProp09	24992	3.5	16328	65.33%	16328	65.33%	16328	65.33%	16328	65.33%	16328	65.33%	16328	65.33%
EuclInvProp12	28458	172.203	18627	65.45%	18627	65.45%	18627	65.45%	18627	65.45%	18627	65.45%	18627	65.45%
EuclInvProp15	27248	3600	18627	68.36%	18627	68.36%	18627	68.36%	18627	68.36%	18627	68.36%	18627	68.36%
EuclUn06	622500	0.828	560101	89.98%	560101	89.98%	560101	89.98%	560101	89.98%	560101	89.98%	560101	89.98%
EuclUn09	994788	3.781	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%
EuclUn12	994788	202.375	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%	987540	99.27%
EuclUn15	1091200	3600	1018160	93.31%	1018160	93.31%	1018160	93.31%	1018160	93.31%	1018160	93.31%	1018160	93.31%
Raidl06	530483	0.484	39850	7.51%			212059	39.97%			39850	7.51%		
Raidl09	598186	12.484	20921	3.50%			132845	22.21%			20921	3.50%		
Raidl12	451005	265.594	20411	4.53%			119659	26.53%			20411	4.53%		
Raidl15	381220	3600	10852	2.85%			86846	22.78%			10852	2.85%		

7.5.2 ME-OCSTP

In the case of formulation III for ME-OCSTP, the family of constraints 5.48 has an exponential number of inequalities because for all $S \subset V$ and for all $i, j \in V : 1, i \in S^c, j \in S$ it requires:

$$f_{ij} \geq R(S) \left(x_{ij} - \sum_{\substack{u \in S, v \in S^c, \\ (u,v) \neq (j,i)}} x_{uv} + x_{vu} \right)$$

For this reason, we can not solve the instances with all the constraints. Hence, our goal is to find a relatively small subset of these constraints that determine the optimal solution, as we have already explained in Section 5.4.

Table 7.6: Results obtained using formulation III for ME-OCSTP.

Formulation III: ME-OCSTP					
Instance	**Optimal/ Best-known***	**Value**	%	**Time** (sec)	**Num Iter**
palm06	180600	180600	100.00%	10	12
palm09	320342	320342	100.00%	726.984	82
palm10	326502	270900	82.97%	>3600	75
palm12	820301*	192554	23.47%	>3600	2
palm16	56700*	0	0.00%	>3600	20
EuclInvProp06	75542	75542	100.00%	18.36	18
EuclInvProp09	134846	95634	70.92%	>3600	175
EuclInvProp12	272025*	52950	19.47%	>3600	14
EuclInvProp15	424402*	0	0.00%	>3600	0
EuclUn06	1149358	1149358	100.00%	19.875	22
EuclUn09	2382568	2232598	93.71%	>3600	137
EuclUn12	3854732*	1357827	35.22%	>3600	24
EuclUn15	13427700*	896724	6.68%	>3600	3
Raidl06	1572645	1572645	100.00%	17.813	28
Raidl09	2117302*	1375828	64.98%	>3600	58
Raidl12	4251531*	971516	22.85%	>3600	42
Raidl15	15799296*	567978	3.59%	>3600	11

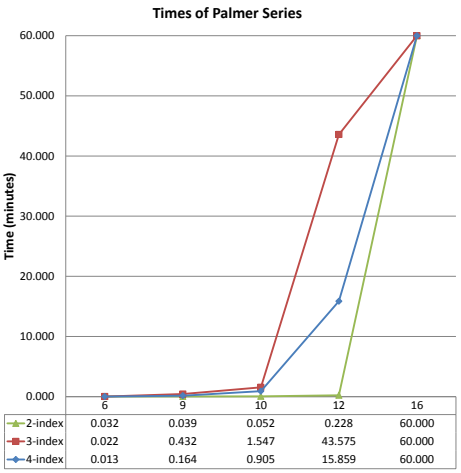
Table 7.6 presents the results obtained with this algorithm to solve the instances. We can see that the number of iterations in a hour decreases very rapidly as the size of the instances increases. Moreover with instances of small

sizes the relaxation produces a good lower bound of the optimal value, while for instances of medium sizes we can not conclude anything on its quality because the optimal value is not available.

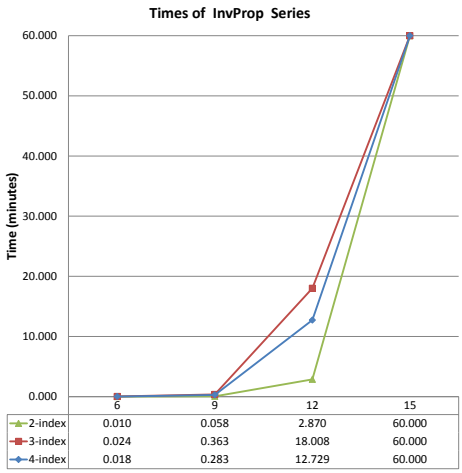
7.6 Comparison between the Three Formulations

In this section we compare the results obtained with the three formulations.

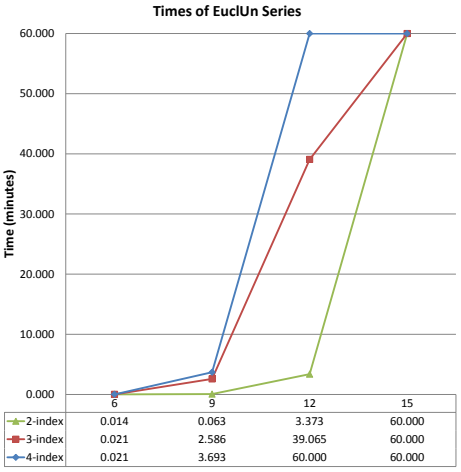
7.6.1 MP-OCSTP



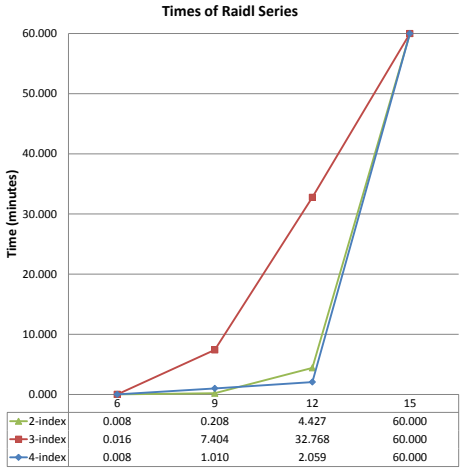
(a) *Computational times Palmer series.*



(b) *Computational times EucInvProp series.*



(c) *Computational times EuclUn series .*



(d) *Computational times Raidl series.*

Figure 7.1: Comparison of computational times to solve MP-OCSTP

First, in Figure 7.1 we show the computational times to solve exactly the MP-OCSTP.
Each graphic corresponds to a single series and it shows the behavior of the computational times for the different formulations with instances with increasing number of nodes. Note that the time limit of 60 minutes is just a censured value which does not necessarily correspond to real computing times.
At a glance, it is evident that with all the formulations the computational times increase with the size of the instances. Moreover with all the instances which satisfy the triangular inequality, the formulation III with 2-index variables is the fastest. In fact, this formulation has a number of variables ($\mathcal{O}(n^2)$) smaller than the other two formulations. Regarding the computational times for the formulations with 3-index and 4-index, non of them systematically outperforms the other one.
Something different happens if we consider the RAIDL series (Figure 7.1d). In fact, in this case the formulations with 2-index and 4-index show a similar behavior, while the computational time using formulation with 3-index variables are worse. Moreover in this case the times increase more rapidly than with other benchmark instances: already with 9 nodes it requires more than 5 minutes.

In Table 7.7 we show for each instance the best result obtained from the relaxation of each of three formulations (in some case adding also a valid inequality) for the MP-OCSTP.
With the PALMER benchmark instances all the relaxations find as a lower bound the optimal value, while with the instances which satisfy the triangular inequality the formulations with 2-index and 3-index are equivalent; in fact the variable that defines the objective function value is the same in both the formulations. Moreover with EUCLUN benchmark instances also the formulation with 4-index is equivalent and the percentage are excellent (around $90-95\%$). Instead the results obtained with EUCINVPROP are inferior and the percentages are around the 70%, although the formulation with 4-index produces a lower bound better than the other two formulations.
When the triangular inequality is not satisfied, the results change considerably. In fact, the results obtained with the formulation with 3-index are not good at all. The formulation with 2-index improves them a little but not enough. Thus, in this case the only formulation that produces good results ($80-85\%$) uses 4-index variables.
In conclusion, to find a good lower bound for MP-OCSTP all three formulations formulations are good. In particular, for solving instances that satisfy the triangular inequality it is recommendable to use the formulation with 2-

Table 7.7: Comparison LP relaxations of the three formulations for the MP-OCSTP.

MP-OCSTP			
Instance	**2-index**	**3-index**	**4-index**
palm06	100.00%	100.00%	100.00%
palm09	100.00%	100.00%	100.00%
palm10	100.00%	100.00%	100.00%
palm12	100.00%	100.00%	100.00%
palm16	100.00%	100.00%	100.00%
EuclInvProp06	76.61%	76.61%	80.14%
EuclInvProp09	65.33%	65.33%	73.25%
EuclInvProp12	65.45%	65.45%	67.07%
EuclInvProp15	68.36%	68.36%	70.55%
EuclUn06	89.98%	89.98%	89.98%
EuclUn09	99.27%	99.27%	99.27%
EuclUn12	99.27%	99.27%	99.27%
EuclUn15	93.31%	93.31%	93.31%
Raidl06	39.97%	7.51%	100.00%
Raidl09	22.21%	3.70%	88.68%
Raidl12	26.53%	4.53%	85.89%
Raidl15	22.78%	2.85%	74.04%

index variables, which has a smaller number of variables, as compared with the others. Otherwise the formulation with 4-index variables is the best.

7.6.2 ME-OCSTP

Since the computational times to solve this problem increase very rapidly with all the formulations, it is not possible compare them. Therefore, in Table 7.8 we only present the values of the LP relaxations of the three formulations of the

three formulations (in some case adding also a valid inequality). We indicate the instances for which the optimal value is not known with a "*".

Table 7.8: Comparison LP relaxations of the three formulations for the ME-OCSTP.

ME-OCSTP			
Instance	**2-index**	**3-index**	**4-index**
palm06	100.00%	42.79%	41.09%
palm09	100.00%	26.80%	26.48%
palm10	82.97%	24.67%	24.31%
palm12*	23.47%	8.89%	9.08%
palm16*	0.00%	53.22%	53.22%
EuclInvProp06	100.00%	21.31%	21.75%
EuclInvProp09	70.92%	11.23%	12.35%
EuclInvProp12*	19.47%	5.67%	6.35%
EuclInvProp15*	0.00%	92.08%	4.18%
EuclUn06	100.00%	28.96%	28.52%
EuclUn09	93.71%	17.52%	18.07%
EuclUn12*	35.22%	10.21%	10.76%
EuclUn15*	6.68%	2.91%	3.09%
Raidl06	100.00%	24.51%	24.51%
Raidl09*	64.98%	15.61%	15.61%
Raidl12*	22.85%	6.48%	6.49%
Raidl15*	3.59%	1.44%	1.52%

It is evident that with small size instances the formulation with 2-index variable produces a lower bound of better quality than the other formulations. Unfortunately to produce a lower bound of the same quality with larger size instances one hour is not enough. In fact, in this case the formulation with 3-index variables is better.

7.7 Heuristic Algorithms for MP-OCSTP

Before presenting the results obtained, we present in Table 7.9 a summery of all the Greedy Algorithms.

Algorithm	**Description**
HE-I	Edges are considered according to their costs (from the smallest to the greatest), and an edge is added only if it does not generate a loop with other edges already chosen.
HE-II	Edges are ordered from the greatest requirement to the smallest one, and an edge is added only if it does not generate a loop with other edges already chosen.
HE-III	Edges are ordered from the greatest product between cost and requirement to the smallest one, and a edge is added only if it does not generate a loop with other edges already chosen.
HE-IV	It's based on the actualization of the matrix $c \cdot r$.
HE-V	It chose the best star-tree.
HE-VI	Edges ordered from the smallest product between the min-cut and the cost to the greatest one, and an edge is added only if it does not generate a loop with other edges already chosen.
HE-VII	Gomory-Hu tree.

Table 7.9: Summary of Greedy Algorithms.

To evaluate the goodness of a heuristic algorithm, we compute the deviation of the objective value $Z(T)$ obtained with with respect to the optimal optimal or best-known value $Z(T^*)$. The deviation is defined:

$$100 \cdot \frac{Z(T) - Z(T^*)}{Z(T^*)}.$$

Therefore, an heuristic algorithm is considered good if the deviation is small. In fact, the deviation is 0 if the algorithm objective value is equal to the optimal value. Note that when the optimal value is not known, the deviation can be strictly negative if the solution obtained with the heuristic outperforms the best solution found by CPLEX.
Moreover, we do not discus the computing times with which we have obtained all the results, because with medium size instances they are influential.

Table 7.10 shows the objective function values and the deviations obtained applying the Greedy Algorithms and Table 7.11 the average calculated for

Table 7.10: Results obtained using all the Greedy Algorithms to solve MP-OCSTP.

Greedy Algorithms: MP-OCSTP															
		HE-I		**HE-II**		**HE-III**		**HE-IV**		**HE-V**		**HE-VI**		**HE-VII**	
Instance	**Optimal**	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation
palm06	133300	133300	0.00	133300	0.00	133300	0.00	133300	0.00	156960	17.75	1.32E+06	892.09	133300	0.00
palm09	141900	141900	0.00	141900	0.00	141900	0.00	141900	0.00	826875	482.72	466257	228.58	172656	21.67
palm10	141900	141900	0.00	141900	0.00	141900	0.00	141900	0.00	826875	482.72	466257	228.58	172656	21.67
palm12	141900	141900	0.00	141900	0.00	141900	0.00	144189	1.61	826875	482.72	392080	176.31	384286	170.81
palm16	106800	106800	0.00	106800	0.00	106800	0.00	327100	206.27	323500	202.90	822600	670.22	106800	0.00
EuclInvProp06	21312	23946	12.36	22512	5.63	156366	633.70	30900	44.99	71280	234.46	23946	12.36	79068	271.00
EuclInvProp09	24992	32433	29.77	25696	2.82	341818	1267.71	31380	25.56	108584	334.48	56790	127.23	148302	493.40
EuclInvProp12	28458	37854	33.02	31428	10.44	341818	1101.13	72864	156.04	108584	281.56	71244	150.35	148302	421.13
EuclInvProp15	27248	31212	14.55	32994	21.09	654414	2301.70	128524	371.68	232403	752.92	31212	14.55	352524	1193.76
EuclUn06	622500	736256	18.27	994500	59.76	994500	59.76	832626	33.76	657885	5.68	665874	6.97	783150	25.81
EuclUn09	994788	1318230	32.51	1702500	71.14	3037970	205.39	1422176	42.96	1012956	1.83	1.69E+06	70.04	1317871	32.48
EuclUn12	994788	1458660	46.63	2065360	107.62	3187200	220.39	1466138	47.38	1012956	1.83	1.34E+06	34.31	1520876	52.88
EuclUn15	1091200	1832840	67.97	4244770	289.00	6411970	487.61	1725080	58.09	1219638	11.77	1.42E+06	30.56	1297890	18.94
Raidl06	530483	585331	10.34	849287	60.10	1705840	221.56	849287	60.10	815475	53.72	585331	10.34	815475	53.72
Raidl09	598186	631609	5.59	1023336	71.07	2989620	399.78	966018	61.49	1066965	78.37	631609	5.59	1130400	88.97
Raidl12	451005	451005	0.00	1867788	314.14	4080190	804.69	983136	117.99	1073932	138.12	451005	0.00	1073932	138.12
Raidl15	381220	454720	19.28	3257056	754.38	4914290	1189.10	1239700	225.19	1200435	214.89	387068	1.53	1387760	264.03

series of the deviations.

Table 7.11: Deviation average for Greedy algorithms for the MP-OCSTP.

Greedy Algorithm Deviation Averages: MP-OCSTP							
Series	**HE-I**	**HE-II**	**HE-III**	**HE-IV**	**HE-V**	**HE-VI**	**HE-VII**
PALMER	0.00	0.00	0.00	0.40	333.76	439.16	42.83
EUCLINVPROP	22.42	9.99	1326.06	108.22	400.85	76.12	594.82
EUCLUN	41.35	131.88	243.29	45.55	5.28	35.47	32.53
RAIDL	8.80	299.92	653.78	116.19	121.28	4.37	136.21

We can observe that with each instance series there is an algorithm outperforming the others. In fact, with PALMER benchmark instances series the algorithms (I, II, III and IV) based on Minimum Spanning Tree are excellent, while with the EUCLINVPROP series the best algorithm is HE-II, but also HE-I and HE-VI produce good results. The HE-I produces good results also with the RAIDL series, but the HE-VI is also very efficient. Instead, the algorithm based on the star-tree is excellent for EUCLUN series.
Finally, it is difficult to say what algorithm is the best, but we can conclude that the algorithms based on the Gomory-Hu tree, the actualization of the weight matrix and the product between the costs and the requirements do not produce reliable upper bounds. This is an unexpected result. In fact the last two algorithms have been thought precisely for this problem.

Now, we comment the results obtained with the local-search algorithm on the trees produced by the greedy heuristics, which are presented in Table 7.12. Moreover in Table 7.13 there are the averages for series of the deviations.
First, we note that this algorithm does not improve considerably the results obtained with HE-I and HE-II, while it improves the others. In particular, if we consider HE-III, the percentages of the deviations with respect to optimal/well-known values have decreased, but however they still remain greater than the percentages obtained with other algorithms. Instead, HE-V produces results very different, depending on whether or not the instance satisfies satisfies the triangular inequality. In fact the percentages are small only in the first case.
The most remarkable aspect is that HE-VI based on the min-cut tree of the original graph produces excellent results with all the instances: with RAIDL benchmark instances the algorithm does not improve the results, but they are

Table 7.12: Results obtained using the Local-Search Algorithm on the trees found with the Greedy Algorithms to solve MP-OCSTP.

Local-Search Algorithm: MP-OCSTP															
		HE-I		**HE-II**		**HE-III**		**HE-IV**		**HE-V**		**HE-VI**		**HE-VII**	
Instance	**Optimal**	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation
palm06	133300	133300	0.00	133300	0.00	133300	0.00	133300	0.00	133300	0.00	133300	0.00	133300	0.00
palm09	141900	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00
palm10	141900	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00
palm12	141900	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00	141900	0.00
palm16	106800	106800	0.00	106800	0.00	106800	0.00	106800	0.00	106800	0.00	106800	0.00	106800	0.00
EuclInvProp06	21312	23946	12.36	22512	5.63	21312	0.00	22512	5.63	22512	5.63	23946	12.36	21312	0.00
EuclInvProp09	24992	32433	29.77	25696	2.82	41768	67.13	31380	25.56	31924	27.74	25696	2.82	32500	30.04
EuclInvProp12	28458	37854	33.02	31428	10.44	42696	50.03	38871	36.59	36414	27.96	32688	14.86	40383	41.90
EuclInvProp15	27248	31212	14.55	32994	21.09	60459	121.88	40152	47.36	34300	25.88	31212	14.55	39144	43.66
EuclUn06	622500	736256	18.27	994500	59.76	994500	59.76	657885	5.68	657885	5.68	622500	0.00	657885	5.68
EuclUn09	994788	1219476	22.59	1702500	71.14	2457542	147.04	1184912	19.11	994788	0.00	1239408	24.59	1099351	10.51
EuclUn12	994788	1359000	36.61	2065356	107.62	3027045	204.29	1389524	39.68	994788	0.00	1336069	34.31	1397724	40.50
EuclUn15	1091200	1832838	67.97	4169440	282.10	3641508	233.72	1621776	48.62	1134476	3.97	1424720	30.56	1219920	11.80
Raidl06	530483	585331	10.34	849287	60.10	849287	60.10	849287	60.10	530483	0.00	585331	10.34	530483	0.00
Raidl09	598186	631609	5.59	1023336	71.07	1433416	139.63	753190	25.91	1066965	78.37	631609	5.59	1023336	71.07
Raidl12	451005	451005	0.00	1867788	314.14	2531958	461.40	695027	54.11	1067116	136.61	451005	0.00	1067116	136.61
Raidl15	381220	454720	19.28	3132600	721.73	4087368	972.18	581440	52.52	1189560	212.04	387068	1.53	1318452	245.85

Table 7.13: Deviation average for Local-Search algorithm for the MP-OCSTP.

Local-Search Algorithm Deviation Averages: MP-OCSTP							
Series	**HE-I**	**HE-II**	**HE-III**	**HE-IV**	**HE-V**	**HE-VI**	**HE-VII**
PALMER	0.00	0.00	0.00	0.00	0.00	0.00	0.00
EUCLINVPROP	22.42	9.99	59.76	16.95	21.80	11.15	28.90
EUCLUN	36.36	130.15	161.20	28.27	2.41	22.37	17.12
RAIDL	8.80	291.76	408.33	48.16	106.75	4.37	113.38

better than those obtained with the combination of the local-search algorithm with any other greedy algorithm. Instead, with the other instances the local-search algorithm enhances the unsuccessful results of this greedy algorithm. Therefore, if we have to suggest a heuristic algorithm to calculate an upper bound of good quality for the MP-OCSTP, the preferred algorithm should be HE-VI followed by local-search algorithm.

7.8 Heuristic Algorithms for ME-OCSTP

The heuristics that we use to solve this problem are the same used to solve the MP-OCSTP (excluded the algorithm based on the weight matrix actualization). Thus, we can see Table 7.9.

Table 7.14 shows the objectives function value and the deviations obtained with the Greedy Algorithms. We can see that HE-I produces good results with each instance and some results are excellent. In fact, the total amount of flow that has to circulate through the tree is constant. Therefore it is likely that activating the cheapest edges we control not only the total communication cost, but also the cost of the most expensive edge of the tree. Despite the good results of this algorithm, with the EUCLUNI benchmark instances the algorithm based on the Star-Tree (HE-V) yields the best results. Moreover when the instances do not satisfy the triangular inequality (RAIDL series), also the algorithm based on the Gormory-Hu Tree (HE-VI) produces excellent results. Note that HE-VI was thought for this specific problem.
Instead the HE-VII does not produce results of poor quality, but for each of the series of benchmark instances there is an algorithm outperforms it.

Table 7.14: Results obtained using all the Greedy Algorithms to solve ME-OCSTP.

Greedy Algorithms: ME-OCSTP													
		HE-I		HE-II		HE-III		HE-V		HE-VI		HE-VII	
Instance	**Optimal Best-known***	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation
palm06	180600	180600	0.00	180600	0.00	180600	0.00	184737	2.29	928356	414.04	180600	0.00
palm09	320342	320342	0.00	320342	0.00	351472	9.72	733084	128.84	524324	63.68	464178	44.90
palm10	326502	332925	1.97	364480	11.63	495256	51.69	766406	134.73	538122	64.81	654069	100.33
palm12	820301*	528165	-35.61	589600	-28.12	589600	-28.12	833050	1.55	572617	-30.19	1033851	26.03
palm16	56700*	119800	111.29	106800	88.36	106800	88.36	322594	468.95	817668	1342.10	106800	88.36
EuclInvProp06	75542	75542	0.00	75542	0.00	244587	223.78	106596	41.11	75542	0.00	115244	52.56
EuclInvProp09	134846	134846	0.00	134846	0.00	539240	299.89	142450	5.64	205898	52.69	198450	47.17
EuclInvProp12	272025*	239687	-11.89	259378	-4.65	744200	173.58	189255	-30.43	365981	34.54	263655	-3.08
EuclInvProp15	424402*	253449	-40.28	271180	-36.10	957090	125.51	278806	-34.31	253449	-40.28	435689	2.66
EuclUn06	1149358	1149358	0.00	2448397	113.02	2448397	113.02	1149358	0.00	1610884	40.16	1573092	36.87
EuclUn09	2382568	4766860	100.07	5395750	126.47	8985960	277.15	2742080	15.09	5220320	119.10	3836800	61.04
EuclUn12	3854732*	6839756	77.44	9457365	145.34	14951874	287.88	3854732	0.00	7005680	81.74	5912180	53.37
EuclUn15	13427700*	9278558	-30.90	24968060	85.94	28267824	110.52	4929133	-63.29	10223140	-23.87	5478807	-59.20
Raidl06	1572645	2265184	44.04	1942360	23.51	3148308	100.19	1764561	12.20	2265184	44.04	1764561	12.20
Raidl09	2117302*	2117302	0.00	3821046	80.47	7350642	247.17	3278718	54.85	2117302	0.00	3278718	54.85
Raidl12	4251531*	2753499	-35.24	9318330	119.18	15673311	268.65	4673556	9.93	2753499	-35.24	4942875	16.26
Raidl15	15799296*	1630811	-89.68	22001640	39.26	20815404	31.75	5400270	-65.82	1630811	-89.68	6166975	-60.97

Now, we comment the results obtained applying the local-search algorithm on the trees produced by the greedy algorithms, which are summarized in Table 7.15.
First, we note that this algorithm does not improve the results obtained with HE-I. In fact, HE-I inserts in the tree the cheapest edges and there is not an another edge with smaller cost that can substitute an edge in the tree. Instead, the local-search algorithm improves the results obtained with all the other algorithms. In particular, it improves considerably the unsatisfactory results of HE-VI and in some cases it produces the best result.
But, what we have to evaluate is the best general heuristic algorithm, which combines the local-search algorithm with a greedy algorithm. Therefore, we think that to solve an instance which satisfies the triangular inequality the best combinations are the heuristics based on the Gomory-Hu tree. Otherwise, HE-II and HE-III are the most efficient.

Table 7.15: Results obtained using the Local-Search Algorithm on the trees found with the Greedy Algorithms to solve ME-OCSTP.

Local-Search Algorithm: ME-OCSTP													
		HE-I		**HE-II**		**HE-III**		**HE-V**		**HE-VI**		**HE-VII**	
Instance	**Optimal Best-known***	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation	Value	Deviation
palm06	180600	180600	0.00	180600	0.00	180600	0.00	180600	0.00	180600	0.00	180600	0.00
palm09	320342	320342	0.00	320342	0.00	320342	0.00	326800	2.02	326800	2.02	320342	0.00
palm10	326502	332925	1.97	326502	0.00	326502	0.00	344000	5.36	344000	5.36	326502	0.00
palm12	820301*	528165	-35.61	528165	-35.61	528165	-35.61	797049	-2.83	528165	-35.61	713489	-13.02
palm16	56700*	119800	111.29	106800	88.36	106800	88.36	244016	330.36	182106	221.17	106800	88.36
EuclInvProp06	75542	75542	0.00	75542	0.00	118844	57.32	92133	21.96	75542	0.00	92133	21.96
EuclInvProp09	134846	134846	0.00	134846	0.00	384540	185.17	142450	5.64	134846	0.00	142450	5.64
EuclInvProp12	272025*	239687	-11.89	239687	-11.89	283620	4.26	189255	-30.43	239687	-11.89	189255	-30.43
EuclInvProp15	424402*	253449	-40.28	253449	-40.28	470844	10.94	251680	-40.70	253449	-40.28	333324	-21.46
EuclUn06	1149358	1149358	0.00	1598280	39.06	1308252	13.82	1149358	0.00	1149358	0.00	1166795	1.52
EuclUn09	2382568	4766860	100.07	3560121	49.42	3062512	28.54	2382568	0.00	4766860	100.07	3121848	31.03
EuclUn12	3854732*	6839756	77.44	6386989	65.69	5486262	42.33	3854732	0.00	6839756	77.44	4629802	20.11
EuclUn15	13427700*	9278558	-30.90	8449840	-37.07	14765171	9.96	4929133	-63.29	9981013	-25.67	4961229	-63.05
Raidl06	1572645	2265184	44.04	1572645	0.00	1572645	0.00	1644928	4.60	2265184	44.04	1644928	4.60
Raidl09	2117302*	2117302	0.00	2453596	15.88	2158436	1.94	2801910	32.33	2117302	0.00	2801910	32.33
Raidl12	4251531*	2753499	-35.24	5249280	23.47	2412300	-43.26	4429425	4.18	2753499	-35.24	3567500	-16.09
Raidl15	15799296*	1630811	-89.68	1630811	-89.68	1833832	-88.39	4955056	-68.64	1630811	-89.68	3502765	-77.83

7.9 Grasp Approach

Since the HE-VI provided good quality solutions, we have developed a randomized version of it, along the line of the Greedy Randomized Adaptive Search Procedure (GRASP).
We have said that in a greedy algorithm, solutions are progressively built. At each iteration, a new element from the set E is incorporated into the partial solution under construction, until a complete feasible solution is obtained. The selection of the next element to be incorporated is determined by the evaluation of all candidate elements according to the greedy evaluation function. The greediness criterion establishes that the local optimum element is selected.
GRASP is a metaheuristic algorithm which consists of iterations made up from successive constructions of a greedy randomized solution and subsequent iterative improvements of it through a local search. The greedy randomized solutions are generated by adding elements to the problem solution set from a list of elements ranked by the greedy function. To obtain variability in the candidate set of greedy solutions, well-ranked candidate elements are often placed in a restricted candidate list (also known as RCL), and chosen at random when building up the solution. Therefore at each iteration of the greedy algorithm we do not insert in the solution the optimum-local element, but we randomly pick one of those $\alpha\%$ best elements.

In the Tables 7.16-7.17 we show the comparison between the results obtained applying HE-VI and the results obtained with the GRASP on the same heuristic. In these cases we have chosen $\alpha = 10\%$ and a number of iteration equal to 50. We have tried to increase the number of iterations, but with our objective functions is useless, because there are not a big variability in the results obtained. We observe that, since the problems are solved 50 times, the computing times increase, but not sufficiently to become relevant.

Moreover, we have tried to change the value of α $(5 - 20\%)$, but the results obtained do not change.

With this procedure with some instances we have obtained a improvement in the upper bound quality. In some cases the deviation decrease and it becomes equal to 0, namely it furnish the optimal value (see RADL series with MP-OCSTP). However, there are instances with which this procedure does not improve the results. In every case, this procedure furnishes good results and it could be used also with other heuristic algorithms because it could improve the upper bounds of poor quality.

Table 7.16: Result obtained with GRASP for the MP-OCSTP.

Grasp: MP-OCSTP					
		HE-VI		**Grasp with HE-VI**	
Instance	**Optimal**	Value	Deviation	Value	Deviation
palm06	133300	133300	0.00	133300	0.00
palm09	141900	141900	0.00	141900	0.00
palm10	141900	141900	0.00	141900	0.00
palm12	141900	141900	0.00	141900	0.00
palm16	106800	106800	0.00	106800	0.00
EuclInvProp06	21312	23946	12.36	23946	12.36
EuclInvProp09	24992	25696	2.82	24992	0.00
EuclInvProp12	28458	32688	14.86	29358	3.16
EuclInvProp15	27248	31212	14.55	28116	3.19
EuclUn06	622500	622500	0.00	622500	0.00
EuclUn09	994788	1239408	24.59	1239408	24.59
EuclUn12	994788	1336069	34.31	1336069	34.31
EuclUn15	1091200	1424720	30.56	1374560	25.97
Raidl06	530483	585331	10.34	585331	10.34
Raidl09	598186	631609	5.59	598186	0.00
Raidl12	451005	451005	0.00	451005	0.00
Raidl15	381220	387068	1.53	381220	0.00

Table 7.17: Result obtained with GRASP for the ME-OCSTP.

Grasp: ME-OCSTP					
		HE-VI		**Grasp with HE-VI**	
Instance	**Optimal/ Best-known***	Value	Deviation	Value	Deviation
palm06	180600	180600	0.00	180600	0.00
palm09	320342	326800	2.02	320342	0.00
palm10	326502	344000	5.36	326502	0.00
palm12	820301*	528165	-35.61	528165	-35.61
palm16	56700*	182106	221.17	119800	111.29
EuclInvProp06	75542	75542	0.00	75542	0.00
EuclInvProp09	134846	134846	0.00	134846	0.00
EuclInvProp12	272025*	239687	-11.89	225920	-16.95
EuclInvProp15	424402*	253449	-40.28	248103	-41.54
EuclUn06	1149358	1149358	0.00	1149358	0.00
EuclUn09	2382568	4766860	100.07	4766860	100.07
EuclUn12	3854732*	6839756	77.44	6839756	77.44
EuclUn15	13427700*	9981013	-25.67	9278558	-30.90
Raidl06	1572645	2265184	44.04	2265184	44.04
Raidl09	2117302*	2117302	0.00	2117302	0.00
Raidl12	4251531*	2753499	-35.24	2753499	-35.24
Raidl15	15799296*	1630811	-89.68	1630811	-89.68

Chapter 8

Concluding Remarks

In this work we have studied two variants of the Optimum Communication Spanning Tree Problem (OCSTP) with different objective functions. In these problems every pair of nodes has a communication requirement which must be satisfied and a cost associated with every direct connection (edge). In the MP-OCSTP the cost of the most expensive path in the spanning tree is minimized. In the ME-OCSTP the objective is to minimize the cost of the most expensive edge in the spanning tree.

First, we have investigated some particular cases to show that the optimal solution has a special structure. In particular, if the costs of the edges are all equal to each other and the communication requirements satisfy an other condition, any solution of the MP-OCSTP is a star-tree.

Then, we have proposed three Mixed Integer Linear Programming (MILP) formulations to solve these problems. The main feature that distinguishes the formulations is the number of indeces of the variable chosen: 4-index, 3-index and 2-index. The computational results obtained on instances with Euclidean and Randomly generated costs show that solving exactly the ME-OCSTP requires a large computing time, even for small-size instances. Therefore, for some instances we have considered the best-known feasible solution to evaluate the lower bounds of the linear relaxations. Instead, the computing times needed for solving the MP-OCSTP are quite moderate: less than one hour for instances with up to 12 nodes and several hours with up to 16 nodes. The formulation with 2-index variables turns out to be more efficient than the other ones.

We have also compared the linear relaxations of all formulations and tried to reinforce them adding some valid inequalities. For the MP-OCSTP instances

that satisfy the triangular inequality it is recommended to use the formulation with 2-index variables, which has a smaller number of variables than the other formulations and provides the same quality results. Instead, if the instances do not satisfy the triangular inequality, the formulation with 4-index variables is the best option, in spite of the high number of variables. For the ME-OCSTP in some case the best lower bound obtained ma be lower than 20% of the value of the best-known solution. Thus, they are unsatisfactory.

Finally, we have presented some greedy heuristics to obtain initial feasible solutions and, for each problem, a local-search algorithm to improve on them. The greedy algorithms are based on the Minimum Spanning Tree or on the star-tree or on the Gomory-Hu tree. The local-search algorithm for the MP-OCSTP tries to add the direct edge for the most expensive path and to delete one of the edges in the generated cycle, while that for the ME-OCSTP tries to delete the most expensive edge and to add one of the edges that connects the two disconnected components. Computational results indicate that the combined greedy-local search heuristic based on the Gomory-Hu Tree provides the best upper bounds for the MP-OCSTP. For the ME-OCSTP, the results vary depending on whether or not the instances satisfy the triangle inequality. In the first case, the heuristics based on the Gomory-Hu tree yield the best solutions, while in the other case those based on the Minimum Spanning Tree are the most efficient. Finally, we have developed a randomized version of the algorithm based on the Gomory-Hu tree and we have seen that in some cases this procedure yields solutions of improved quality.

Since the formulations for the ME-OCSTP do not produce a lower bound of good quality, future research could aim at finding some valid inequalities that strengthen them and thus improve the value of the linear relaxation. Moreover, an alternative could be to consider a new formulation based on the column generation. An additional possible direction for further research is the study of decomposition solution methods that allow to handle medium and large size instances with the 4-index formulations.
As to the heuristic methods, apart from randomized versions of the other greedy algorithms, the study of more sophisticated heuristic algorithms based on the exploration of large neighborhoods in a flexible fashion (Variable Neighborhood Search) seems promising as it could produce improved solutions. Another option could be to develop a meta-heuristic, such as for instance tabu-search, to try to escape from local minima.

Bibliography

[1] Ahuja, R.K., and V.V.S Murty, *Exact and Heuristic Algorithms for the Optimum Communication Spanning Tree Problem*, Transportation Science 21 (1987), 163-170.

[2] Contreras, I., *Network Hub Location - Models, Algorithms, and Related Problems*, Ph.D thesis, Universitat Politècnica de Catalunya, 2009.

[3] Contreras, I., E. Fernández, and A. Marín, *Lagrangian bounds for the optimum communication spanning tree problem*, TOP 18 (2010), 140-157.

[4] Contreras, I., E. Fernández, and A. Marín, *Tight bounds from a path based formulation for the tree of hub location problem*, Computers & Operations Research 36 (2009), 3117-3127.

[5] Contreras, I., E. Fernández, and A. Marín, *The Tree of Hubs Location Problem*, Computers & Operations Research 202 (2010), 390-340.

[6] Feng D., and R. Doolittle, *Progressive sequence alignment as a prerequisite to correct phylogenetic trees*, J. Molecular Evol. 25 (1987), pp. 351-360

[7] Fernández, E., C. Luna, and G. Reinelt, *A compact formulation for the optimum communication spanning tree problem*, International Symposium on Mathematical Programming (ISMP 2012), Berlin, 2012.

[8] Fernández, E., C. Luna, A. Hildenbrandt, G. Reinelt, and S. Wiesberg, *A Flow Formulation for the Optimum Communication Spanning Tree*, Electronic Notes in Discrete Mathematics 41 (2013), 85-92.

[9] Fischetti, M., G. Lancia, and P.Serafini, *Exact Algorithms for Minimum Routing Cost Trees* Networks 39 (2002), 161-173.

[10] Gusfield, D., *Very simple methods for all pairs network flow analysis*, SIAM J. COMPUT. 19 (1990), 143-155.

[11] Hu, T.C., *Optimum Communication Spanning Trees*, SIAM J. Comput. 3 1974, 188-195.

[12] Johnson, D.S., and J.K. Lenstra, and A.H.G. Rinnooy Kan, *The complexity of the Network Design Problem*, Networks 8 (1978), 279-285.

[13] Kruskal J. B., *On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem*, Proceedings of the American Mathematical Society 7 (1956), 48-50.

[14] Rothlauf, F., *On Optimal Communication Spanning Tree Problem*, Operation Research 57 (2009), 413-425.

[15] Wang L., and T. Jiang, *On the complexity of multiple sequence alignment*, J. Comput. Biol. 1 (1994), 337-348.

[16] Wu B. Y., G. Lancia, V. Bafna, K-M. Chao, R. Ravi, and C. Y. Tang, *A polynomial-time approximation scheme for minimal routing cost spanning trees*, SIAM J. Comput 29 (1999), 761-778.