The Optimum Communication Spanning Tree Problem

Properties, models and algorithms

Author: Carlos Luna-Mota

Advisor: Elena Fernández

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament d'Estadística i Investigació Operativa

The research presented in this PhD dissertation has been partially supported by the MTM2009-14039-C06-05 research project of the Spanish Ministry of Education and Science, MTM 2012-36163-C06-05 research project of the Spanish Ministry of Economy and Competitiveness and the grant BES-2010-034835 from the Spanish Ministry of Education and Science.

Keywords: Optimum Communication Spanning Tree Problem, Network Design, Spanning Trees, Combinatorial Optimization.

MSC2010: 68M10, 90-02, 90B10, 90B18, 90C05, 90C10, 90C11, 90C27, 90C35, 90C59.

UNESCO: 120700, 120707, 120709, 120710.

I went to the woods because
I wished to live deliberately,
to front only the essential facts
of life, and see if I could not
learn what it had to teach,
and not, when I came to die,
discover that I had not lived

Henry David Thoreau

Acknowledgements

I am in debt with all persons who made possible this PhD thesis but I want to specially acknowledge the help that I received from my advisor, Elena Fernández. I cannot stress enough how grateful I am for her commitment and for the trust that she placed in me. Elena has been, and will always be, a mentor and an example for me.

I am also very grateful to Ivan Contreras, who preceded me on the study of the OCSTP. Ivan has been not only a tireless collaborator, but also a kind friend without whom this project would never have been possible.

I wish to acknowledge the warm hospitality that I received during my research stages at Heidelberg University and at Montréal University. My mentors there, Gerhard Reinelt and Jean-François Cordeau, gifted me with their vast knowledge and inspired many of the results of this dissertation.

I am thankful to my colleagues from the Departament d'Estadística i Investigació Operativa, who provided a great working environment and were always ready to offer their help and sympathy.

Finally, I wish to acknowledge the support that I received from my parents, Bartolomé and María Rosa, and the infinite patience, encouragement and love that my wife, Núria, has devoted to me all these years. *Sense tu, no hauria valgut la pena fer aquest viatge.*

Carlos Luna-Mota Barcelona, December 10, 2015

Contents

Chapter 1

Introduction

In the last decades, networks have become increasingly important in our lives. From public transport to the wireless connectivity of our smartphones, we use many systems closely related to transportation, telecommunication and computer networks on a daily basis. Network design and network analysis tools are thus required to manage the increasing complexity of our interconnected world.

The design of efficient networks, in particular, may be of special interest not only to telecommunication or transportation companies seeking to increase their benefits but also to all of us, since their efficiency will translate in great savings of time and energy in our daily activities.

Not all network design problems, however, share the same notion of efficiency. In some contexts, such as electric power networks, it is fundamental to minimize the total length of the network edges, since both, the construction cost and the transportation cost of the network, increase with each additional kilometer of high tension line. In these contexts, optimal solutions are Minimum Spanning Trees or, more generally, Minimum Steiner Trees.

In other situations, however, communication requirements between origin/destination pairs are given, and the objective is to minimize the operational cost of the network. The communication cost between an origin/destination pair, depends both on the communication requirement and on the distance on the network between the vertices of the pair. Hence, if the only criterion is the minimization of communication costs, the resulting network would be the union of the shortest

paths between all pairs. This will likely involve a quadratic number of connections, which might increase the construction and maintenance cost of the network unnecessarily.

There are many efficiency criteria that look for a compromise between the construction and operational costs of a network. The criterion used by the **Optimum Communication Spanning Tree Problem** (hereafter, OCSTP) is to find a spanning tree of minimum operational cost. Among all possible topologies for a network that connect n locations, spanning trees minimize the number of established connections, imposing a tight limit on the construction costs. And since the network topology is used to control the construction costs, we are free to choose the spanning tree that provides the smallest communication costs, even if this means that many long edges must be built (which is rarely the case).

The OCSTP was originally introduced by Hu (1974). Despite its apparent simplicity, the OCSTP turns out to be a very challenging combinatorial optimization problem. Johnson et al. (1978) proved that finding an OCST is $\mathcal{NP}$-hard, even if all origin/destination pairs have the same communication requirement. Moreover, unless $\mathcal{NP} = \mathcal{P}$, no polynomial time approximation scheme exists since the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard (Papadimitriou and Yannakakis, 1988). In comparison, other spanning tree optimization problems, such as the Minimum Spanning Tree problem, can be solved in polynomial time and are often left as exercise in many computer science courses (see, for example, Cormen et al., 2009).

Even though it is very difficult to find OCSTs in general, there are two particular cases, identified by Hu (1974), which can be solved in polynomial time. The first one is the Optimum Requirement Spanning Tree Problem where, in the original graph, the distance between all pairs of vertices is the same. In this case, an optimal solution is given by a Gomory-Hu Min-Cut tree of an associated network. In the other particular case, called the Optimum Distance Spanning Tree Problem, all pairs of vertices have the same communication requirements and, under some additional assumptions, it has star-shaped trees as optimal solutions.

Apart from being a theoretically challenging problem, the OCSTP can be applied to several real-world problems regarding the design of communication networks where it is necessary to impose a tree topology to avoid the routing decisions or the synchronization issues associated with cycles. The OCSTP is also relevant when the price of building a given network does not depend on the total

length of that network as much as on the amount of connections established, as it is the case in many virtual networks. Finally, designing infrastructures using OCST ensures that each of the edges of the network will have a high degree of use, which is a desirable quality in transportation networks with high building costs, such as a train network. Even in the cases where we consider no restriction relative to the network topology, the OCSTP can be used to identify the critical subnetwork that must be built first to ensure global connectivity and reasonable communication costs: Since all connected networks contain a spanning tree as a subgraph, the use of an OCST is justified as an efficient initial inversion when the allocated budget (in terms of time, workforce or economical resources) is not enough to build the whole planned network at once.

Besides its specific applications regarding the design of optimum communication networks, the OCSTP appears as a subproblem in some Hub Location Problems such as the Tree-of-Hubs Location Problem (Contreras et al., 2010b; Contreras and Fernández, 2012), that can be polynomially transformed into an OCSTP when the set of hubs and the allocation pattern of nodes to hubs are known. Additionally, the particular case of the OCSTP where all communication requirements are equal is used in computational biology to find optimal alignments of genetic sequences (Wu et al., 2000c; Fischetti et al., 2002).

Our research on the OCSTP, summarized in this dissertation, provides new exact and heuristic algorithms for general instances of the OCSTP. In particular, we describe an efficient Branch & Cut algorithm based on an enhanced Benders reformulation of the problem. This exact algorithm consistently outperforms CPLEX with any known formulation of the problem and is able to provide optimal solution for previously unsolved instances from the literature. In addition, we have developed several new Mixed Integer Linear Programming formulations with interesting theoretical properties and improved a previously known formulation for the problem with new families of valid cuts. Regarding heuristic algorithms, we have extended the most widely used local search algorithm for the OCSTP. We also developed a novel Divide & Conquer constructive heuristic and proposed a new family of spanning tree neighborhoods along with an innovative exploration strategy that fully exploits the structural relationships between such neighborhoods. Finally, we propose two new combinatorial lower bounds based on the inherent characteristics of the the OCSTP.

These results have given rise to the publication Fernández et al. (2013c) and the communications Contreras et al. (2015a,b); Fernández and Luna-Mota (2012a,b, 2014) and Fernández et al. (2013a,b).

The remainder of this document is organized as follows: Chapter 2 includes the mathematical foundations of our research and settles the notation used through this dissertation. Chapter 3 introduces the OCSTP and reviews the previous literature on this subject and discusses several linear models. Chapters 4 and 5 describe the heuristic and exact algorithms that we developed to solve general instances of the OCSTP as well as the computational experiences that we used to evaluate them. Finally, Chapter 6 includes the conclusions of our research and what we consider the most promising lines of future research on the subject.

Chapter 2

Mathematical preliminaries

2.1 Graph Theory

The following conventions will be used throughout this thesis:

G always denotes a simple, undirected **graph**, without loops or multi-edges, with **vertex set** $V = \{1, 2, \ldots, n\}$ and **edge set** $E \subseteq \{\{i, j\} \mid i, j \in V; i \neq j\}$. The **degree** of a vertex is the number of edges incidents with that vertex.

For a subset of vertices $S \subseteq V$ we denote the set of edges with both ends on S as $E(S) = \{\{i, j\} \in E \mid i \in S \text{ and } j \in S\}$ and the set of edges with one end on S and the other end on $\bar{S} = V \setminus S$ as $\delta(S) = E \setminus \big(E(S) \cup E(\bar{S})\big)$. The pair $\{S, \bar{S}\}$ is called a **cut** of G. Indistinctively, the set of edges with one end on S and the other end on $\bar{S}$ will also be referred to as a cut.

When we consider directed arcs (instead of undirected edges) the notation will be: $A = \{(i, j) \mid i, j \in V; i \neq j\}$ (with parentheses instead of braces to indicate that the order of the elements does matter).

We say that G' is a **subgraph** of G if $V' \subseteq V$ and $E' \subseteq E$. When $V' = V$, G' is called a **spanning subgraph** of G.

A simple **path** is a sequence of edges of the form $(\{i_1, j_1\}, \ldots, \ \{i_m, j_m\})$ with $j_k = i_{k+1}\ \forall k \in \{1, 2, \ldots, m-1\}$ and with any given vertex label appearing in at most two edges. A closed path ($j_m = i_1$) is called a **cycle**. If arcs are used instead of edges, we obtain directed paths and directed cycles respectively.

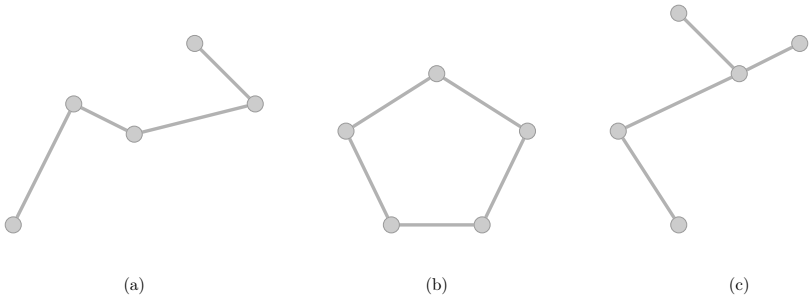


Figure 2.1: A path (a), a cycle (b) and a tree (c).

If we assign lengths to the edges of the graph we can define the **distance** d_{ij} between two different vertices as the length of the shortest path of G connecting both vertices. The distances defined thereby satisfy the triangle inequality:

$$d_{ik} \leq d_{ij} + d_{jk} \tag{2.1}$$

If we assign capacities to the edges of the graph we can define the **capacity** of a cut $\{S, \bar{S}\}$ as the sum of the capacities of the edges that traverse the cut.

The Max-Flow-Min-Cut theorem (Ford and Fulkerson, 1956) states that the maximum amount of flow f_{ij} that we can send between two different vertices of a capacitated graph is equal to the smallest capacity amongst all the cuts of G that separate both vertices. As a corollary of that theorem we can deduce that the maximum flows defined thereby satisfy the following inequality for all $k \in V$:

$$f_{ik} \geq \min(f_{ij}, f_{jk}) \tag{2.2}$$

A graph $G = (V, E)$ is **connected** if there exists a path $P_{ij} \subseteq E$ between every pair of vertices. A graph G is **acyclic** if does not contain any cycle. Acyclic subgraphs are also called **forests**. A connected forest is called a **tree**.

K_n denotes the **complete graph** with $E = \{\{i, j\} \in V \times V \mid i \neq j\}$. The set of all spanning trees of K_n are denoted as $\mathcal{T}_n$ and its elements ($T \in \mathcal{T}_n$) are referred simply as *trees*. When we need to work with smaller (non-spanning) trees we denote them as *subtrees*.

2.1.1 Spanning trees

Spanning trees are rich in mathematical properties. To give an idea of this richness we present the following equivalent statements (Eisner, 1997; Lawler, 2001):

1. T is a *spanning tree* of G.
2. T is a connected and acyclic spanning subgraph of G.
3. T is a connected spanning subgraph of G with $n-1$ edges.
4. T is an acyclic subgraph of G with $n-1$ edges.
5. T is a connected spanning subgraph of G and removing an edge from T will disconnect it.
6. T is an acyclic subgraph of G and adding any edge to T will create a cycle.
7. There is a unique path over T connecting each pair of vertices of G.

In a tree, vertices of degree 1 are called **leaves** whereas the rest of the vertices are denoted **interior** vertices. All trees have between 2 and $n-1$ leaves. **Paths** and **stars** are special cases of trees, characterized by having an extremal number of leaves. Paths are trees with only 2 leaves whereas stars are characterized by having $n-1$ leaves.

Spanning trees and cuts:

Two cuts of G, $\{S, \bar{S}\}$ and $\{Q, \bar{Q}\}$, are said to cross each other if and only if each of the four sets $S \cap Q$, $S \cap \bar{Q}$, $\bar{S} \cap Q$ and $\bar{S} \cap \bar{Q}$ contain at least one vertex. A set of two or more cuts of G are said to be non-crossing if no two of them cross each other.

Lemma 2.1.1 (Hu, 1974)**.** *Each spanning tree T uniquely determines $n-1$ non-crossing cuts and each set of $n-1$ non-crossing cuts of a graph G uniquely determines a spanning tree.*

Representing Spanning Trees:

The set $\mathcal{T}_n$ of spanning trees of K_n grows quickly as n increases. However, each tree T is linear in size and, hence, it may be desirable to represent it using just $\mathcal{O}(n)$ space.

Several such representations are available in the literature, being the most commonly used ones the adjacency lists representation, the edge-set representation and the predecessor list representation (Cormen et al., 2009):

Adjacency lists: A tree T is represented as a dictionary of n lists, with $list[k]$ containing the indices of all the neighbors of k in T. Since T contains $n-1$ edges, the total amount of space used is $2(n-1)$. This representation can be useful in many heuristic algorithms that need to perform searches over the set of neighbors of a given vertex frequently.

Edge-set list: A tree T is represented as a set of $n-1$ edges sorted by a given suitable criterion. Since T contains $n-1$ edges and each edge can be represented by a pair of vertex indices, the total amount of space used is $2(n-1)$. This representation has been used as a genetic code in evolutionary heuristic algorithms, with each edge being a *gene* of the tree.

Predecessor list: A vertex of T is chosen and a Depth First Search is performed from this node, recording the vertex used to reach each vertex through an edge during the search. Since T is connected, the search will reach all the vertices; and since T is acyclic, the *predecessor* of a given node is uniquely determined by the starting point of the search. Therefore, we can represent T using a list of n vertex indices, with the kth element being the predecessor of vertex k in the search (or a NULL marker if k was the starting point, which is usually chosen as the last element of the list and omitted from it). The predecessor list representation of a tree can be thought of as a compact version of the edge-set list representation. Unlike the previous representations, the predecessor list representation is unable to represent arbitrary directed graphs or graphs with more than n edges and therefore may only be used to represent trees, forests and cycles in general.

Checking if a given edge is in T may require a linear search if we use any of the compact representations described above and, thus, another data structure

should be used when this operation is required frequently. The adjacency matrix representation offers a trade-off between the time required to check if a given edge is in the tree and the amount of space required to store it:

> **Adjacency matrix:** A tree T is represented as a binary matrix where the cell (i, j) is equal to 1 if and only if there is and edge between i and j in T. This representation has interesting theoretical properties in the spectral graph theory field and may be useful to represent the design variables of a Linear Programming formulation of a network design problem.

Finally, it is worth noting that, none of the above representations is limited to representing spanning trees, not even the predecessor list representation, which is highly related to the structural properties of the spanning trees. This means that other graphs different from spanning trees may be represented with them and sometimes the process to determine if these graphs effectively belong to $\mathcal{T}_n$ may require as much as $\mathcal{O}(n)$ operations. It would be interesting to find a compact data structure that would be able to represent spanning trees only (preferably uniquely).

Cayley's formula (Cayley, 1889) for the number of different labeled spanning trees of n vertices states that:

$$|\mathcal{T}_n| \;=\; n^{n-2} \tag{2.3}$$

Therefore, the set $\mathcal{C}_n$ of words of length $n-2$ on the alphabet $\{1, 2, \ldots, n\}$, hereafter **Cayley strings**, may be an obvious candidate as a data structure to represent trees. Note that the predecessor list data structure uses words of lengths $n-1$. Any bijection between $\mathcal{C}_n$ and $\mathcal{T}_n$ is called a **Cayley code**. Most of these $(n^{n-2})!$ possible bijections do not have any structural feature that justifies its use. There are, however, some remarkable Cayley codes with practical applications:

> **Prüfer code:** (Prüfer, 1918) In order to represent a tree T with vertices labeled by the elements of $\{1, 2, \ldots, n\}$, repeat $n-2$ times the following procedure: select the leaf of T with smallest label, annotate the label of its only neighbor and delete the selected leaf from T. The resulting sequence of $n-2$ labels uniquely determines T and the process can be reversed efficiently. The Prüfer code has good structural properties (for example, the degree of

each vertex of T is equal to the number of times this vertex appears in the code plus one) but fails to provide *high locality* (Gottlieb et al., 2001), i.e. Cayley strings that differ in a single character can be transformed by the Prüfer decoding algorithm into trees that have no edges in common. As a consequence the Prüfer code is mainly used as a proof for Cayley's formula nowadays.

Dandelion code: (Picciotto, 1999) This Cayley code, which can be seen as a refinement of the predecessor list representation, possesses the best known locality bound among the set of Cayley codes. In particular, the Dandelion Code guarantees that two Cayley strings differing in just 1 character will be mapped to spanning trees that differ in at most 5 edges independently of n. Furthermore, as $n \to \infty$, the expected number of edges that will change in a tree if we change a single character of its corresponding Cayley string is 1. Both properties make the Dandelion code the best candidate to represent trees in evolutionary algorithms. In Section 4.2.2 we will present an extension of the Dandelion Code that allows us to represent neighborhoods of $\mathcal{T}_n$. Finally, the $\mathcal{O}(n)$ encoding and decoding algorithms for the Dandelion code proposed by Paulden and Smith (2006) can be found in the Appendix A.

The main drawback of using Cayley codes as representations of trees is that it is difficult to establish a direct relationship between the characters of the Cayley string and the edge (or edges) that these characters represent. Even with the exceptional locality bound of the Dandelion code, it is often impossible to predict how a given tree will change if we slightly modify its Cayley string. This means that there is no workaround to use the $\mathcal{O}(n)$ procedure of encoding the tree, applying the modifications, and then decoding it into a new Cayley string. Determining how the Cayley string of a given tree changes if we change a single edge of that tree is equally difficult. Therefore, Cayley codes should only be used in contexts where using this $\mathcal{O}(n)$ encoding-transforming-decoding procedure does not constitute a cumbersome overhead.

2.1.2 Some Notable trees

As we have already seen, a tree can be understood as an acyclic set of $n-1$ edges or as a non-crossing set of $n-1$ cuts. Both definitions are equivalent but lead to very different points of view. The former is commonly associated with edge lengths whereas the latter is usually preferred when we have to deal with cut capacities.

In this section we will review two well-known families of trees that appear in many combinatorial optimization problems. Both families are apparently unrelated but, in fact, there is a clear parallelism in their properties. This parallelism will be evident again when we study the OCSTP and, thus, it is justified to study the properties of Minimum Spanning Trees and Min-Cut Trees using a common framework.

Minimum Spanning Trees:

Let G be an undirected graph with a cost (length) function $c : E \to \mathbb{R}$ associated with its edges. A Minimum Spanning Tree (MST) of G is a spanning tree of G that minimizes $\sum_{e \in T} c_e$. It may not be unique if several edges have the same length. An excellent survey of the many polynomial time algorithms that find MSTs as well as their structural properties can be found in Eisner (1997).

The main properties of the MSTs are:

Cut Property of MSTs: Let T^* be an MST with respect to c. Let e^* be one of its edges, c^* its length and $\{S, \bar{S}\}$ the cut associated with $T^* \setminus \{e^*\}$. Then for any other edge e traversing $\{S, \bar{S}\}$, the length of e is such that $c^* \leq c_e$.

Cycle Property of MSTs: Let T^* be the MST with respect to c and $\bar{e} \in E \setminus T^*$. Let $\{e_1, e_2, \ldots, e_k\}$ be the unique path over T^* that connects the endpoints of $\bar{e}$ and $c_1, c_2, \ldots, c_k$ their lengths. Then the length of $\bar{e}$ is such that $\bar{c} \geq c_i \quad \forall\, i \in \{1, 2, \ldots, k\}$.

The Cut Property has been fundamental in the development of many algorithms that find MSTs, whereas the Cycle Property can be used to easily proof that a given edge does not belong to an MST. A less known property of MSTs that will be relevant for our study of the OCSTP is:

Length Sequence Property of MST: Let T^* be the MST with respect to c and T be any other tree. Let also $c_1^* \leq c_2^* \leq \ldots \leq c_{n-1}^*$ and $c_1 \leq c_2 \leq \ldots \leq c_{n-1}$ be the respective sorted sequences of lengths of their edges. Then, $c_i^* \leq c_i \quad \forall\, i \in \{1, 2, \ldots, n-1\}$.

This latter property can be easily derived from Kruskal's algorithm for constructing MST (Kruskal, 1956).

Min-Cut Trees:

Let G be an undirected graph with a capacity function $w : E \to \mathbb{R}$ associated with its edges and f_{ij} the matrix of maximum $ij-$flows associated with w. Gomory and Hu (1961) proved that it always exists a (non-necessarily unique) weighted tree, called the Min-Cut Tree (MCT) with the following properties:

Cut Property of MCTs: Let T^* be an MCT with respect to w. Let e^* be one of its edges, f^* its associated maximum flow and $\{S, \bar{S}\}$ the cut associated with $T^* \setminus \{e^*\}$. Then $f^* = \max_{e \in \delta(S)} (f_e)$.

Cycle Property of MCTs: Let T^* be an MCT with respect to w and $\bar{e} \in E \setminus T^*$. Let $\{e_1, e_2, \ldots, e_k\}$ be the unique path over T^* that connects the endpoints of e and $f_1, f_2, \ldots, f_k$ their associated maximum flows. Then the maximum flow associated with $\bar{e}$ is such that $\bar{f} = \min_{i \in 1 \ldots k} (f_i)$.

Both properties are derived from the fact that MCTs are maximum spanning trees with respect to the matrix f_{ij}. However, these properties are often presented in a slightly different and more useful form:

Equivalent Flow Property of MCTs: For every pair of vertices $\{i, j\}$ of a capacitated graph G with capacity function w, the maximum amount of flow that can be sent between i and j over G and over an MCT of G is the same.

Equivalent Cut Property of MCTs: For every pair of vertices $\{i, j\}$ of a capacitated graph G with capacity function w, and for every T^*, MCT of G, there is an edge e^* in the unique path of T^* that connects both vertices such that the cut associated with $T^* \setminus \{e^*\}$ is a cut of minimum capacity separating i and j.

The Equivalent Flow Property characterizes MCTs as the subgraphs of G with minimum total capacity that allow the same functionality as G itself. It should be noted that, in particular, this means that at most $n-1$ different values can be present in the f_{ij} matrix and suggests that $n-1$ max-flows queries should suffice to compute the whole matrix (instead of the $\binom{n}{2}$ required by a brute force approach). This intuition is confirmed by the algorithms presented by Gomory and Hu (1961) and Gusfield (1990).

The Equivalent Cut Property is stronger than the Equivalent Flow Property in the sense that there are Equivalent Flow trees that are not Equivalent Cut trees. This is also the most useful property of MCTs, since it allows us to store all the minimum cuts G in a compact data structure.

2.2 Mixed Integer Linear Programming

Linear Programming is a technique for the optimization of minimization (or maximization) problems that can be expressed in terms of a **linear objective function** and a set of **linear inequality constraints**.

Generic Linear Programming formulation:

$$
\begin{aligned}
[\mathbf{P}] \qquad \text{Min} \quad & cx && (2.4a)\\
\text{s.t.} \quad & Ax \geq b && (2.4b)\\
& x \geq 0 && (2.4c)
\end{aligned}
$$

where x is a vector of **decision variables** and the coefficients matrix A as well as the independent terms vector b contain real values and have the appropriate dimensions.

Several algorithms can be used to obtain optimal solutions for such problems, being the Simplex family one of the most popular and efficient ways to find them in practice. Geometrically, the **feasible region** defined by the sets of inequalities $Ax \geq b$ and $x \geq 0$ can be interpreted as a convex polytope in $\mathbb{R}^n$ (with $n = |x|$). Any point of this polytope is considered a **feasible solution** and optimizing a Linear Programming problem is to find the feasible solution that minimizes the objective function. If the polytope is non-empty and bounded, the problem of finding and optimal solutions can be reduced to the problem of finding a vertex of the polytope that minimizes the objective function and the Primal Simplex algorithm exploits this fact by finding an arbitrary initial vertex and iteratively pivoting to better vertices traversing edges of the polytope in a greedy fashion until a local optimum is reached (which will also be a global optimum since both, the polytope and the objective function, are linear.
When some or all the variables are required to be integer-valued, we talk about Mixed Integer Linear Programming (MILP) formulations. MILP problems are $\mathcal{NP}$-hard in general and more sophisticated algorithms are required to find optimal solutions. The **Branch & Bound** algorithm is one of the simplest approaches and serves as a base for many other MILP algorithms. To solve a MILP problem, the Branch & Bound algorithms finds first the optimal solution, x^*, of its **linear**

relaxation (i.e. the same problem without the integrality constraints). If all the variables that must be integer-valued in the original problem are integer-valued in x^* the algorithm stops and return x^*. Otherwise, a fractional-valued variable x_i is selected and two new problems are defined by adding, respectively, the constraints: $x_i \geq \lceil x_i^* \rceil$ and $x_i \leq \lfloor x_i^* \rfloor$. These two problems are solved recursively and their optimal solutions are compared to obtain the global optimal solution of the original problem. It must be noted that this enumerative procedure terminates in a finite (although potentially exponential) number of steps. In order to speed-up the Branch & Bound algorithm, each time that an integer solution is found, the value of the best integer solution found so far (the **incumbent**) is updated. The incumbent is an upper bound on the value of the global optimal solution and can be used to prune problems whose linear relaxation value is equal or worse than the incumbent, avoiding the explicit exploration of large areas of the feasible region.

In our research, we make use of several well-known results related to MILP formulations, including the capacity of modern software suites to solve continuous linear models efficiently and to apply a general purpose Branch & Bound algorithm to solve moderate size instances of integer models. Other advanced implementation features, such as the dynamic addition of valid inequalities in a Branch & Cut framework or the possibility to define sophisticated branching rules, are also fundamental to develop a competitive exact algorithm for the OCSTP. All of these techniques are considered standard and a comprehensive exposition of them can be found in advanced textbooks on the subject (such as the excellent Wolsey, 1998). Below we describe, however, a well-known technique, called Benders decomposition, given the relevance that it has in the research that we have developed.

2.2.1 Benders Decomposition

Decomposition methods are techniques that allow to break huge linear models into simpler subproblems that can be solved independently and then linked via some additional conditions. As such, they often establish a trade-off between the size and complexity of the problems to be solved and the number of them. The most usual case consists of transforming a single MILP formulation into a series of smaller and simpler subproblems that will be solved iteratively until they converge to an optimal solution to the initial problem.

Benders decomposition (Benders, 1962) is one of the most popular of such decomposition methods and can be summarized as follows:

Let us consider the following generic MILP formulation of a minimization problem, where x and y are vectors of integer and continuous decision variables, respectively, and the matrices A, Q and R, as well as the vectors b, c, d, and s, contain real values and have the appropriate dimensions:

$$
\begin{aligned}
[\mathbf{P}] \qquad \text{Min} \quad & cx + dy && \text{(2.5a)}\\
\text{s.t.} \quad & Qx + Ry \;\geq\; s && \text{(2.5b)}\\
& Ax \;\geq\; b && \text{(2.5c)}\\
& x \;\geq\; 0 \text{ and integer} && \text{(2.5d)}\\
& y \;\geq\; 0 && \text{(2.5e)}
\end{aligned}
$$

Now, we observe that for any given feasible value of the integer variables, $\bar{x}$, the restricted subproblem is just a continuous linear program in the y variables and can be solved efficiently in general.

Primal Subproblem:

$$
\begin{aligned}
[\mathbf{PS}(\bar{\mathbf{x}})] \qquad \text{Min} \quad & dy && \text{(2.6a)}\\
\text{s.t.} \quad & Ry \;\geq\; (s - Q\bar{x}) && \text{(2.6b)}\\
& y \;\geq\; 0 && \text{(2.6c)}
\end{aligned}
$$

Furthermore, since $\mathbf{PS}(\bar{\mathbf{x}})$ is a continuous linear program, for the values of $\bar{x}$ which make $\mathbf{PS}(\bar{\mathbf{x}})$ feasible, its dual counterpart $\mathbf{DS}(\bar{\mathbf{x}})$ exists and the optimum value for both is the same.

Dual Subproblem:

$$
\begin{aligned}
[\mathbf{DS}(\bar{\mathbf{x}})] \qquad \text{Max} \quad & \pi(s - Q\bar{x}) && \text{(2.7a)}\\
\text{s.t.} \quad & \pi R \;\leq\; d && \text{(2.7b)}\\
& \pi \;\geq\; 0 && \text{(2.7c)}
\end{aligned}
$$

More specifically, if y^* is an optimal solution to $\mathbf{PS}(\bar{\mathbf{x}})$, then $\mathbf{DS}(\bar{\mathbf{x}})$ has an optimal solution π^* and $dy^* = \pi^*(s - Q\bar{x})$. In contrast, if $\mathbf{PS}(\bar{\mathbf{x}})$ is infeasible, then $\mathbf{DS}(\bar{\mathbf{x}})$ is unbounded and there exists an extreme ray $\tilde{\pi}$ such that $\tilde{\pi}(s - Qx) \geq 0$.

Now we must note that the feasible region of $\mathbf{DS}(\bar{\mathbf{x}})$ does not depend upon the value of $\bar{x}$. This is a key observation that allows to express $\mathbf{P}$ as two nested optimization problems.

$$\min_{x \in \left\{ \begin{smallmatrix} Ax \geq b \\ x \geq 0 \end{smallmatrix} \right\}} \left\{ cx + \max_{\pi \in \left\{ \begin{smallmatrix} \pi R \leq d \\ \pi \geq 0 \end{smallmatrix} \right\}} \{\pi(s - Qx)\} \right\} \tag{2.8}$$

Let $\mathbb{DP} = \{\pi \geq 0 \mid \pi R \leq d\}$ be the polyhedron that all dual subproblems $\mathbf{DS}(\bar{\mathbf{x}})$ have in common, $\mathcal{V}$ its set of vertices and $\mathcal{R}$ its set of extreme rays. Now we can use an auxiliary continuous variable z to obtain the Benders reformulation of $\mathbf{P}$.

Benders reformulation:

$$\begin{aligned}
[\mathbf{M}_\infty] \quad \text{Min} \quad & cx + z & & & (2.9a)\\
\text{s.t.} \quad & Ax \geq b & & & (2.9b)\\
& z \geq \pi^*(s - Qx) & & \forall\, \pi^* \in \mathcal{V} & (2.9c)\\
& 0 \geq \tilde{\pi}(s - Qx) & & \forall\, \tilde{\pi} \in \mathcal{R} & (2.9d)\\
& x \geq 0 \text{ and integer} & & & (2.9e)\\
& z \geq 0 & & & (2.9f)
\end{aligned}$$

Constraints (2.9c) and (2.9d) are called **Benders optimality cuts (BOC)** and **Benders feasibility cuts (BFC)**, respectively. This latter family, however, can be omitted if the primal subproblem can be proven to be always feasible.

The reader may wonder whether or not it is convenient to project out the set of continous variables y at expenses of exchanging a polynomial set of constraints (2.5b) by two potentially exponential sets of constraints such as (2.9c) and (2.9d). However, it is fundamental to observe that for many formulations only a few constraints of these families will be needed to find an optimal (x^*, z^*) pair, and that we can relax them to get a simplified Master Problem that may be easy to solve despite having integrality constraints.

Initial Master Problem:

$$
\begin{aligned}
[\mathbf{M_0}] \quad \text{Min} \quad & cx + z && && (2.10a)\\
\text{s.t.} \quad & Ax \geq b && && (2.10b)\\
& x \geq 0 \text{ and integer} && && (2.10c)\\
& z \geq 0 && && (2.10d)
\end{aligned}
$$

Once we obtain an initial solution $(\bar{x}^0, \bar{z}^0)$ we can evaluate the feasibility of $\bar{x}^0$ to obtain either a Benders optimality cut (if the primal subproblem is feasible), or a Benders feasibility cut (if the primal subproblem turns out to be infeasible). In either case we obtain a new constraint, which can be added to $\mathbf{M^0}$ to obtain $\mathbf{M^1}$ and a new optimal solution $(\bar{x}^1, \bar{z}^1)$. This iterative process continues until a solution appears twice, indicating that the algorithm has converged to the global optimum solution: $(\bar{x}^{k-1}, \bar{z}^{k-1}) = (\bar{x}^k, \bar{z}^k) = (x^*, z^*)$.

Master Problem after t iterations:

$$
\begin{aligned}
[\mathbf{M_t}] \quad \text{Min} \quad & cx + z && && (2.11a)\\
\text{s.t.} \quad & Ax \geq b && && (2.11b)\\
& z \geq \pi^*(s - Qx) && \forall\, \pi_1^*, \ldots, \pi_k^* \in \mathcal{V}_t && (2.11c)\\
& 0 \geq \tilde{\pi}(s - Qx) && \forall\, \tilde{\pi}_{k+1}, \ldots, \tilde{\pi}_t \in \mathcal{R}_t && (2.11d)\\
& x \geq 0 \text{ and integer} && && (2.11e)\\
& z \geq 0 && && (2.11f)
\end{aligned}
$$

It is important to note that, in each iteration, the value of the objective function of the Master Problem provides a valid lower bound on the optimum value whereas evaluating any feasible solution through the primal or dual Subproblems provides a valid upper bound. This means that a tight optimality gap can be computed in each iteration and the algorithm may be stopped prematurely if the best solution found is within an acceptable range of the optimum solution.

It is also use fundamental to note that for many formulations the huge number of continuous variables y makes impractical to solve the original formulation directly, which justifies the use of Benders decomposition to project them out.

In fact, even the continuos Subproblem may be difficult to solve due to their size. Fortunately, in many formulations associated with network design problems, the subproblem decomposes in a set of smaller problems that can be solved independently (Magnanti and Wong, 1981). This is useful, since the amount of time and memory required to solve them at once may be several orders of magnitude bigger.

Another technique that speeds up the process consists in solving just a single Master Problem, considering (2.9c) and (2.9d) as cutting planes and separating them dynamically each time an integer solution is found. In fact, if the subproblem can be solved for fractional values of x, cuts can be generated at each node of the Branch & Bound tree. This variant, which is just an alternative way to manage the high number of constraints of the Benders reformulation, is often called *Branch & Cut Benders Decomposition* to differentiate it from the standard *Iterative Benders Decomposition*.

Finally, when the subproblem shows degeneracy (as it is usual in network design problems), many different optimality cuts may be generated from the same Master Problem solution. In those cases, the number of iterations needed to converge greatly depends on the criterion used to select the cut that will be added to the Master Problem in each iteration. Several methods have been proposed in the literature, being Magnanti and Wong (1981) (with the improvements proposed by Papadakos, 2008) one of the most successful ones. The Magnanti-Wong method (and the Papadakos variant) consists in solving an alternative Subproblem that requires the use of *core points* (points in the relative interior of the domain defined by the constraints $Ax \geq b$). This method ensures that the family of cuts generated is Pareto-optimal (no cut of the family is dominated by another one, where $z \geq \pi_1^*(s - Qx)$ is said to dominate $z \geq \pi_2^*(s - Qx)$ if $\pi_1^*(s-Qx) \geq \pi_2^*(s-Qx)$ with strict inequality for at least one point of the Master Problem domain) and usually converges to the optimal solution in fewer iterations than the naïve Benders approach.

Chapter 3

The OCSTP

3.1 Problem definition

The Optimum Communication Spanning Tree Problem (OCSTP), originally introduced by Hu (1974), is formally defined as follows. Let $G = (V, E)$ be a connected undirected graph with $n = |V|$ vertices and $c : E \to \mathbb{R}^+$ a cost function over the edges of G. Let $R \subset V \times V$ be a set of communication requirements and $w : R \to \mathbb{R}^+$ its corresponding weighting function. The solution space of the OCSTP is the set of all spanning trees of G. For a given spanning tree T of G,

The **communication cost over T of a pair of vertices** $\{o, d\}$ is defined as the cost (length) with respect to c of the (unique) path in T that connects o and d, multiplied by the communication requirement between the pair: $dist^T(o, d)w^{od}$ and the **total communication cost of** T is the sum of the communication costs over T of all pairs of vertices:

$$\sum_{\{o,d\} \in R} w^{od} \cdot dist^T(o, d)$$

The OCSTP is to find a spanning tree of G of minimum total communication cost. Equivalently, the OCSTP can be defined as the problem of finding the spanning tree of G that minimizes the weighted average of path lengths between pairs of vertices (using w^{od} as weights) or the weighted average of flows traversing its edges (using c_{ij} as weights).

3.1.1 Additional assumptions

In our study of the OCSTP we assume that $G = K_n$, with $V = \{1, 2, \ldots, n\}$ and $E = \{\{i, j\} \mid i \in V,\ j \in V \text{ and } i \neq j\}$. This is an assumption commonly used in many network design problems because it is natural to evaluate the desirability of any possible connection in the design phase, even if this connection is not directly available in the real world (in which case G is completed with additional edges whose lengths are determined by the shortest path connecting both endpoints).

Regarding the lengths of the edges of G we have already stated that they are symmetrical ($c_{ij} = c_{ji}$), and non-negative ($c_{ij} \geq 0$) with $c_{ij} = 0 \Leftrightarrow i = j$ (if $i \neq j$ and $c_{ij} = 0$ we can assume that $\{i, j\}$ will be part of an optimal solution and transform the instance into an equivalent smaller instance where vertices i and j are substituted by vertex (ij) with $c_{(ij)k} = \min\{c_{ik}, c_{jk}\}$ and $w^{(ij)k} = w^{ik} + w^{jk}$ for all $k \in V \setminus \{i, j\}$).

Finally, with respect to the communication requirements we assume that they are non-negative ($w^{od} \geq 0$) and that they induce a connected subgraph of G (i.e. the graph $\{V, E'\}$ with $E' = \{\{o, d\} \mid w^{od} > 0\}$ is connected). Any instance that does not satisfy this latter assumption can be decomposed in as many independent (smaller) subproblems as connected components the graph induced by R has.

Although it is not necessary for the theoretical part of our study, it is convenient to mention that all the values that define a given instance (the number n, the lengths c_{ij} and the weights w^{od}) must be representable in the memory of a computer. Hence, the assumption that lengths and weights may belong to the realm of the non-negative real numbers $\mathbb{R}^+$ should be strengthened a little bit by admitting only rational lengths and weights. But once we define this finite set of values as rational numbers there is no theoretical impediment to multiply them by the minimum common multiple of their denominators, obtaining an equivalent instance defined by integer values. However, as the efficiency of some algorithms may be affected by the magnitude of the values that define a given instance, this change of units should not be applied blindly and the quantitative results presented in this dissertation should be interpreted taking into account the range of values allowed for the instances described here.

3.1.2 Notable particular cases of the OCSTP

There are two well-known particular cases of the OCSTP which have been identified as polynomially solvable by Hu (1974) and deserve our attention.

Optimum Requirement Spanning Tree Problem (ORSTP): It is the particular case of the OCSTP where all $\binom{n}{2}$ possible pairs of vertices of G are connected with an edge of the same length (we can assume without lost of generality that $c_{ij} = 1 \quad \forall\, i \neq j$). If we express the objective function of the OCSTP as a function of the flow f_e traversing the edges of T:

$$\sum_{e \in T} c_e f_e,$$

it is clear that, for the ORSTP, we can factor out the edge lengths c_e and ignore them altogether since they are all equal. But the objective function $\sum_{e \in T} f_e$ is precisely what MCTs minimize and, thus, all MCTs are optimal solutions to the ORSTP.

Optimum Distance Spanning Tree Problem (ODSTP): It is the particular case of the OCSTP where all $\binom{n}{2}$ possible pairs of vertices of G have the same communication requirements (we can assume without lost of generality that $w^{od} = 1 \quad \forall\, o \neq d$). For this particular case, Hu (1974) shows that there is an optimal solution among the n possible star-shaped trees of G, provided that the following technical condition over the cost function c_e is satisfied:

- There exists $0 \leq \lambda \leq \frac{n-2}{2n-2}$ such that for all possible triplets $\{i, j, k\}$ of different vertices of G the following inequality holds: $c_{ij} + \lambda c_{jk} \geq c_{ki}$.

Note that allowing $\lambda = 1$ in the previous condition is equivalent to the triangle inequality. But, in fact, the above condition implies that $\lambda \to 1/2$ from below as $n \to \infty$, which is more restrictive and can be understood intuitively as imposing that all edges of G are of similar length. On the other hand, if all edge lengths are equal, the previous condition can be satisfied for all values of n by selecting $\lambda = 0$.

3.1.3 State-of-the-Art

As mentioned, the OCSTP was formally introduced by Hu (1974). In this seminal paper, Hu identifies the two particular cases described above and proposes polinomial time algorithms for them. The OCSTP was soon classified as $\mathcal{NP}$-hard by Johnson et al. (1978). Moreover, Papadimitriou and Yannakakis (1988) showed that the problem is $\mathcal{MAX}$ $\mathcal{SNP}$-hard, which implies that unless $\mathcal{NP} = \mathcal{P}$ no polynomial time approximation scheme exists for the OCSTP. More recently, Rothlauf (2009b) and also Steitz and Rothlauf (2008, 2009, 2012a) studied the properties of optimal solutions to the OCSTP and determined that they tend to include short edges (as MSTs do) and tend to have few vertices of high degree near the graph center and a large percentage of leaves (as stars do). Other properties, such as the relative orientation of the edges with respect to the geometric barycenter of the vertices, are equally interesting but can not be applied to instances whose vertex coordinates are not available.

The OCSTP appears naturally in Network Design Problems where the communication requirements of the network are known in advance and the network topology must not include cycles for economic reasons (such as minimizing the construction costs) or for practical reasons (such as avoiding routing decisions). The OCSTP appears also as a subproblem in the Tree-of-Hubs Location Problem (Contreras et al., 2010b; Contreras and Fernández, 2012) when the set of hubs and the node-to-hub allocation pattern are fixed. Another practical application of the OCSTP is reported in (Wu et al., 2000c; Fischetti et al., 2002) where the authors use a particular case of the OCSTP to find optimal alignments of genetic sequences by defining a complete graph whose vertices are genetic sequences and whose edges are weighted by the value of edit distance between their endpoints and then finding the spanning tree that minimizes the unweighed sum of path lengths between pairs of vertices (which is known as the sum-of-pairs objective in the field of computational biology).

Due to the challenging nature of the OCSTP, even for moderate size instances, few exact methods have been proposed in the literature. The first one, from Ahuja and Murty (1987), consists in a Branch & Bound enumerative procedure where lower bounds are obtained using lower planes. The authors reported competitive solution times for sparse instances with up to 40 vertices (here *sparse* means that the underlying graph G has about 10% of the edges of the complete graph K_n).

Regarding mathematical programming formulations, Fischetti et al. (2002) proposed a MILP formulation for a particular case of the OCSTP where all communication requirements are equal. Rothlauf (2007) presented a MILP formulation for the general case of the OCSTP that is able to solve instances with up to 12 vertices. Two years later, Contreras (2009) presented another integer programming formulation which, enforced with additional valid inequalities, was able to produce optimal solutions for instances with up to 25 vertices in reasonable computational times. Contreras et al. (2010a) proposed a Lagrangian relaxation which produced good lower and upper bounds for instances with up to 50 vertices. Fernández et al. (2013c) developed the improved flow-based formulation that we describe in Section 3.3. Finally, Tilk and Irnich (2015) presented a combined column-and-row generation algorithm for the OCSTP that produces good results for sparse instances with up to 30 vertices.

Ahuja and Murty (1987) also proposed a two phase heuristic for the OCSTP, which first builds a spanning tree and then tries to improve it examining its 1-edge-exchange neighborhood until a local optimum spanning tree is found. The local search used for the tree-improvement phase remains as one of the most flexible and practical algorithms to deal with arbitrarily large instances and has greatly influenced the research on heuristic algorithms for the OCSTP.

Fischer (2007) proposes a variant of the Ahuja-Murty local search that, at each step, only considers the most promising subset candidate edges to be included in the tree in order to increase the overall speed of the search when dealing with bigger instances. Later, Fischer and Merz (2007) use this enhanced local search in combination with an evolutionary algorithm, to obtain a memetic algorithm that outperforms other evolutionary approaches (such as Palmer and Kershenbaum, 1994; Li and Bouchebaba, 2000; Soak, 2006) for large escale instances. Wolf and Merz (2010) describe an alternative exploration strategy for the 1-edge-exchange neighborhood which, applied to the special case where all communication requirements are equal, has the same worst case $\mathcal{O}(n^3)$ cost than the Ahuja-Murty local search, but which improves its expected case complexity to just $\mathcal{O}(n^2 \log n)$.

Other heuristic approaches to the OCSTP include approximation algorithms for different restricted cases of the OCSTP. Peleg (1997); Reshef and Peleg (1998); Wu et al. (2000a,b,c, 2002) propose a $\mathcal{O}(n^5)$, 1.577-approximation algorithm for the Product-Requirement Communication Spanning Tree Problem (where

$w^{ij} = q(i) \cdot q(j)$ for a fixed function $q : V \to \mathbb{R}$), a $\mathcal{O}(n^3)$ 2-approximation algorithm for the Sum-Requirement Communication Spanning Tree Problem (where $w^{ij} = q(i) + q(j)$) and a $\mathcal{O}(n^3)$ 1.577-approximation algorithm for the Minimum Routing Cost Spanning Tree Problem (where all $w^{ij} = 1$). Wu (2002); Ravelo and Ferreira (2015) treated the p-source particular case, where only a small subset of vertices sends information through the network. It is often difficult or impractical to apply algorithms based on particular cases to general instances of the OCSTP, Sharma (2006), however, proposed two pseudo-polynomial algorithms based on particular cases that can be applied to the general instances to obtain solutions that are locally optimal with respect to the 1-edge-exchange neighborhood.

Finally, several evolutionary algorithms have been proposed in Rothlauf (2006, 2009a,b); Steitz and Rothlauf (2008, 2009, 2012a,b) but they were designed to study and test the characteristics of optimal solutions to the OCSTP and none of them outperforms the other heuristic algorithms cited above.

3.2 Upper and lower bounds

3.2.1 Upper bounds

Being the OCSTP a minimization problem, the evaluation of any feasible solution (i.e. any spanning tree) provides a valid upper bound. Therefore, high quality heuristic solutions can be very useful algorithmically in enumeration-based methods, as they can be used to prune many nodes of the search tree.

As mentioned in Section 3.1.2, the OCSTP has two particular cases whose optimal solution can be found in polynomial time. Since the computational burden required to find those solutions is negligible, it is always advisable to build them as a pre-process step and use the best of their communication costs as an initial upper bound. In particular, we can build the MCT with respect to the communication requirements matrix w and compute the n star-shaped trees to quickly find a good heuristic solution that covers both polynomially solvable particular cases of the OCSTP. The MST with respect to the cost matrix c, which minimizes the construction cost of the network instead of its operational cost, may also be of interest and can be built in $\mathcal{O}(|E|\alpha(|E|,|V|))$ steps (where $\alpha(m,n)$ is the functional inverse of Ackermann's function).

In addition to these known spanning trees, borrowed from other computational problems, there are some constructive heuristics specifically developed for the OCSTP that produce high quality solutions at the expenses of a bigger computational effort. Such constructive heuristics (along with the previously mentioned ones) can be further enhanced by means of a local search algorithm. Several of such heuristics are reviewed in Chapter 4.

3.2.2 Lower bounds

Being the OCSTP a minimization problem, the linear relaxation of any integer formulation provides a valid lower bound. In an enumerative framework, the smallest value of all unexplored nodes provides such lower bound at any moment.

Global lower bounds are used to estimate the optimality gap during the execution and at the end of any exact algorithm, but may also be useful in a preprocessing step to evaluate the *a priori* difficulty of a given instance. The following combinatorial lower bounds can be computed for this purpose.

The Second-shortest-path lower bounds

It is obvious that the objective function value, z, is lower bounded by the corresponding weighted sum of shortest path lengths:

$$z \geq \sum_{r \in R} w^r D_G(o^r, d^r) \tag{3.1}$$

where $D_G(i,j)$ is the length of the shortest path of G connecting i and j.

For a euclidean instance, whose cost matrix c satisifies the triangle inequality $c_{ik} \leq c_{ij} + c_{jk} \forall i,j,k \in V$, this **shortest-path lower bound** is equivalent to assume that all communication requirements are satisfied using a direct connection but, since the optimal solution only contains $n-1$ edges, at least $|R| - (n-1)$ communication requirements must be served trough a route that is, at least, as long as $D^2_G(o^r d^r)$, the length of the second-shortest-path. From this fact the following global **second-shortest-path lower bound** (Reinelt, 2013) can be derived:

$$z \geq \min_{T \in \mathcal{T}_n} \left\{ \sum_{\{i,j\} \in T} w^{ij} D_G(i,j) + \sum_{\{i,j\} \notin T} w^{ij} D^2_G(i,j) \right\} \tag{3.2}$$

Which can be restated as:

$$z \geq \sum_{\{i,j\} \in E} w^{ij} D^2_G(i,j) + MST(c^*) \tag{3.3}$$

where the $MST(c^*)$ is the cost of a Minimum Spanning Tree with respect to the cost matrix c^* with $c^*_{ij} = w^{ij}(D_G(i,j) - D^2_G(i,j))$.

Note that, since $w^r \geq 0 \quad \forall\, r \in R$ and $D^2_G(i,j) \geq D_G(i,j) \quad \forall\, \{i,j\} \in E$, c^* only contains non-positive values and, therefore, $MST(c^*)$ accounts for the $n-1$ edges with the biggest difference between $D_G(i,j)$ and $D^2_G(i,j)$ that form a spanning tree (i.e. the ones that provide the biggest savings, and thus the smallest lower bound, under the condition of defining a spanning tree).

This lower bound can be particularized to smaller subsets of vertices $S \subset V$:

$$\sum_{\{i,j\}\subset S} z^{ij} \;=\; \sum_{\{i,j\}\subset S} w^{ij} d^{ij} \;\geq\; \min_{T\in\mathcal{T}(S)} \left\{ \sum_{\{i,j\}\in T} w^{ij} D_G(i,j) + \sum_{\{i,j\}\notin T} w^{ij} D_G^2(i,j) \right\} \tag{3.4}$$

where $\mathcal{T}(S)$ is the set of all subtrees spanning the vertices of S.

For small subsets S, (3.4) inequalities can be separated by inspection and can be included in any linear formulation that uses the z^r or the d^r variables to compute communication costs or distances (respectively). As the size of S increases, however, these cuts become weaker so it is rarely useful to consider subsets of 5 or more vertices.

The MST-MCT global lower bound

Let T^{MCT} be an MCT with respect to the communication request matrix w, T^{MST} an MST with respect to the cost matrix c and T^* an OCST with respect to w and c.

Let $f_1 \leq f_2 \leq \cdots \leq f_{n-1}$ be the equivalent flows that traverse the edges of T^{MCT} and $f_1^* \leq f_2^* \leq \cdots \leq f_{n-1}^*$ the equivalent flows that traverse the edges of T^*, sorted in increasing order.

Let $c_1 \leq c_2 \leq \cdots \leq c_{n-1}$ be the lengths of the edges of T^{MST} and $c_1^* \leq c_2^* \leq \cdots \leq c_{n-1}^*$ the lengths of the edges of T^*, sorted in increasing order.

As stated in Section 2.1.2, the lengths sequence of T^{MST} is the lexicographically smallest length sequence among all spanning trees of G, so, in particular:

$$c_i \;\leq\; c_i^* \qquad \forall i = 1..n-1 \tag{3.5}$$

Equivalently, an MCT minimizes the sum of edge-flows among all equivalent-flow subgraphs of G that satisfy the communication requirements of w. Moreover, all edge-flows of T^{MCT} are associated with a minimum cut of G with respect to w. Thus, we can use the greedy algorithm from Matroid theory (see Lawler, 2001) to find T^{MCT}, which implies that its sorted equivalent flow sequence is also lexicographically minimal among all equivalent flow sequences of spanning trees of G.

Again, in particular:

$$f_i \leq f_i^* \qquad \forall i = 1..n-1 \tag{3.6}$$

Thus, if $\mathcal{S}_{n-1}$ denotes the set of all permutations of the indices $\{1, 2, \ldots, n-1\}$, the following expression provides a valid lower bound for the OCSTP:

$$z \geq \min_{\sigma \in \mathcal{S}_{n-1}} \left\{ \sum_{i=1}^{n-1} c_i f_{\sigma(i)} \right\} \tag{3.7}$$

In fact, the rearrangement inequality (see Bulajich Manfrino et al., 2009) shows that the permutation that minimizes such sum of products is, precisely, $\sigma(i) = n - i$. Therefore:

$$z \geq \sum_{i=1}^{n-1} c_i f_{n-i} \tag{3.8}$$

is a valid global lower bound for the OCSTP.

Computational experience

In Table 3.1 we report the average optimality gap obtained by the lower bounds described above. Columns **LB1**, **LB2** and **LB3** contain, respectively, the average gaps obtained with the Shortest-path lower bound (3.1), the Second-shortest-path lower bound (3.3) and the MST-MCT lower bound (3.8).

We observe that the Shortest-path lower bound and the Second-shortest-path lower bound obtain similar results. This is due to the fact that the length of the shortest-paths and the length of the second-shortest-paths are often similar for these sets of instances. We also observe that the Second-shortest-path lower bound is consistently tighter than the MST-MCT lower bound in general but we have found instances where the MST-MCT algorithm provides a better lower bound. All of these combinatorial lower bounds have been computed in less than 0.1 seconds per instance.

Set	**|V|**	**LB1**	**LB2**	**LB3**
Con	10	11.37%	9.41%	46.91%
Con	20	16.52%	16.18%	58.56%
Con	30	17.68%	17.42%	59.11%
Con	40	19.92%	19.43%	62.6%
Con	50	21.57%	21.35%	67.17%
RE	15	12.5%	11.34%	42.22%
RE	20	16.41%	16.14%	53.77%
RE	25	15.74%	15.33%	53.18%
RE	30	17.74%	17.54%	58.3%
RE	35	18.98%	18.68%	58.61%
RE	40	22.43%	22.2%	61.42%
RE	50	20.34%	20.11%	60.66%
RE	60	21.43%	21.16%	62.51%
RE	70	22.23%	21.93%	62.82%
RE	80	25.59%	25.34%	65.24%
RE	90	25.27%	24.99%	66.02%
RE	100	27.14%	26.83%	66.74%

Table 3.1: Comparison of global lower bounds.

3.3 MILP formulations for the OCSTP

By using linear models of the OCSTP we can take advantage of the rich theory developed to solve such models (cutting plane methods, lifting procedures, decomposition methods, etc.) and the wide variety of solvers available nowadays.

To the best of our knowledge, only two Linear integer programming formulations have been used in the literature to solve medium size instances of the OCSTP. We started our research studying the path-based and flow-based formulations described in Contreras (2009). The path-based formulation for the OCSTP provides better linear relaxation bounds but uses a large number of variables and constraints, which limits the size of the instances that we are able to load in memory using standard hardware. The flow-based formulation can be obtained by aggregating some subsets of variables of the path based formulation and, as a consequence, it provides weaker linear relaxation bounds. Although we proposed several new families of valid cuts that reinforce the flow-based formulation, the computational results obtained applying them to medium size instances did not produce noticeable improvements.

After our initial exploration of these formulations, we developed a new linear formulation based on rooted trees. To the best of our knowledge this is the most compact formulation for the OCSTP, requiring just $\mathcal{O}(n^2)$ variables (which is optimal) and $\mathcal{O}(n^3)$ constraints. Even after the addition of several families of valid cuts, the rooted tree formulation remains too weak to solve medium size instances too and, even if it remains theoretically interesting, we focused again on the path-based formulation, exploiting techniques that allow us to deal with its large number of variables and constraints (see Chapter 5).

3.3.1 Variables and objective functions

Modeling Spanning Trees: Since our final objective is to find OCSTs and not only their communication cost, it is necessary for any given formulation of the OCSTP to include some binary design variables, x_{ij}, which are equal to 1 if the edge $\{i, j\}$ is in the solution and 0 otherwise.

Spanning trees appear often in combinatorial optimization problems and can be modeled in many ways using the x_{ij} design variables (see Magnanti and Wolsey, 1995 for an excellent reference on the subject). The most popular spanning tree

linear integer programming models are based on these alternative definitions:

1. T is a connected spanning subgraph of G with $n-1$ edges.
2. T is an acyclic subgraph of G with $n-1$ edges.

The first one leads to:

$$\sum_{\{i,j\}\in E} x_{ij} = |V| - 1 \tag{3.9a}$$

$$\sum_{\{i,j\}\in\delta(S)} x_{ij} \geq 1 \qquad \forall\, S \subsetneq V \tag{3.9b}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, \{i,j\} \in E \tag{3.9c}$$

whereas the second one is usually translated as:

$$\sum_{\{i,j\}\in E} x_{ij} = |V| - 1 \tag{3.10a}$$

$$\sum_{\{i,j\}\in E(S)} x_{ij} \leq |S| - 1 \qquad \forall\, S \subsetneq V \tag{3.10b}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, \{i,j\} \in E \tag{3.10c}$$

Both formulations include a cardinality constraint, ensuring that exactly $n-1$ edges are selected. On the other hand, the Cut-Set Inequalities (CSI)(3.9b) require that all possible cuts are crossed by at least 1 edge whereas the Subtour Elimination Constraints (SEC) (3.10b) prevent any cycle to be formed (since a cycle through k vertices requires k edges). Both families of constraints are equivalent if we impose integrality, but the linear relaxation of the second one is stronger than the linear relaxation of the first one (as shown in Magnanti and Wolsey, 1995) and SEC are thus preferred.

Note that SEC is a family of constraints of exponential size on the number of vertices and, therefore, they should be separated dynamically instead of including them at once in the initial formulation. Fortunately, there is a well-known $\mathcal{O}(n^4)$ time procedure for separating SEC based on the MCT with respect to the capacity vector x_{ij} (see, for example, Korte and Vygen, 2007). If all x_{ij} are integer, there is an even faster $\mathcal{O}(n^2)$ SEC-separation procedure based on the connected components induced by x_{ij}.

Objective functions: Since the objective function of the OCSTP cannot be expressed as a linear function of the design variables x_{ij}, it is necessary to include some additional variables (and constraints) in any formulation in order to be able to compute the communication cost of the resulting network.

The original definition of the OCSTP provided by Hu (1974) defines the objective to minimize (z) as a sum of the communication costs (z^r) and then states that each of them can be computed as the product of some (known) communication requirements (w^r) and distances (d^r):

$$z \;=\; \sum_{r \in R} z^r \;=\; \sum_{r \in R} w^r d^r \tag{3.11}$$

Although not impossible (we will see examples for all of them below), it is often complicated to model the continuous variables z, z^r or d^r in terms of the design variables x_{ij}. A common approach in network design is to model the directed paths that will be used to satisfy each communication requirements with the directed binary variables y^r_{ij}:

$$\sum_{r \in R} w^r d^r \;=\; \sum_{r \in R} w^r \sum_{\{i,j\} \in E} c_{ij} y^r_{ij} \tag{3.12}$$

The above approach is often unpractical since it makes use of a large amount of variables (up to $\mathcal{O}(n^4)$). A traditional workaround to this problem is to aggregate the path variables that share a common origin using variables f^o_{ij} that model flows over the edges of T:

$$\sum_{\{i,j\} \in E} c_{ij} \sum_{r \in R} w^r y^r_{ij} \;=\; \sum_{\{i,j\} \in E} c_{ij} \sum_{o:(o,d) \in R} f^o_{ij} \tag{3.13}$$

If we further aggregate the f^o_{ij} variables associated with a given edge:

$$\sum_{\{i,j\} \in E} c_{ij} \sum_{o:(o,d) \in R} f^o_{ij} \;=\; \sum_{\{i,j\} \in E} c_{ij} f_{ij} \tag{3.14}$$

we obtain an expression of the objective function where the f_{ij} variables can be interpreted as the minimum edge capacity needed to satisfy all communication requirements and the c_{ij} represent the cost of adding a unit of capacity to the edge $\{i, j\}$.

Under this latter interpretation the OCSTP becomes a special case of the Multiterminal Network Synthesis Problem (Gomory and Hu, 1961).

3.3.2 Path-based and flow-based formulations

In addition to the binary design variables x_{ij}, the path-based formulation for the OCSTP uses the continuous variables y^r_{ij} to determine if arc (i,j) is in the path used to satisfy the communication requirement r. These paths are modeled using the design variables as capacities (3.15f) and then imposing the classical flow conservation constraints ((3.15c)–(3.15e)), as if one unit of flow must be sent between each origin-destination pair.

Path-based formulation:

$$\text{Min} \quad \sum_{r\in R}\sum_{(i,j)\in A} w^r c_{ij} y^r_{ij} \tag{3.15a}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} x_{ij} \;=\; |V|-1 \tag{3.15b}$$

$$\sum_{i:(i,d^r)\in A} y^r_{id^r} \;=\; 1 \qquad \forall\, r\in R \tag{3.15c}$$

$$\sum_{i:(i,j)\in A} y^r_{ij} - \sum_{k:(j,k)\in A} y^r_{jk} \;=\; 0 \qquad \forall\, r\in R \quad \forall\, j\in V\setminus\{o^r,d^r\} \tag{3.15d}$$

$$-\sum_{k:(o^r,k)\in A} y^r_{o^r k} = -1 \qquad \forall\, r\in R \tag{3.15e}$$

$$y^r_{ij} + y^r_{ji} \;\le\; x_{ij} \qquad \forall\, r\in R \quad \forall\, \{i,j\}\in E \tag{3.15f}$$

$$x_{ij}\in\{0,1\} \qquad \forall\, \{i,j\}\in E \tag{3.15g}$$

$$y^r_{ij} \;\ge\; 0 \qquad \forall\, r\in R \quad \forall\, (i,j)\in A \tag{3.15h}$$

Note that, as long as the communication requirements induce a connected subgraph, it is not necessary to include CSI or SEC, since the connectivity requirements are imposed through the constraints that require that the underlying network is able to send a unit of flow between such pairs of vertices. Note also that it is not necessary to impose the integrality on the y^r_{ij} variables since the design variables x_{ij} necessarily form a spanning tree and in such kind of network there is a unique path connecting each pair of vertices.

Among the previously known linear formulations for the OCSTP, the path-based formulation (3.15) provides the tightest linear relaxation bounds. However, the large number of variables and constraints used by this formulation (up to $\mathcal{O}(n^4)$) precludes its direct application even for moderate size instances when used with a commercial solver. A straightforward solution to this problem consists in aggregating the y_{ij}^{od} that share a common origin, which leads to the flow-based formulation.

Indeed, flows associated with different communication requirements should not cancel each other out and, thus, it is necessary to represent them with different variables. Nevertheless, it is not necessary to define a set of flow variables for each communication requirement, since flows with the same origin circulating on a spanning tree cannot traverse a given edge in opposite directions, given that such networks are acyclic. The flow-based formulation presented above makes use of this property to reduce the number of variables from $\mathcal{O}(n^4)$ to $\mathcal{O}(n^3)$ by aggregating them by origins.

Flow-based Formulation:

$$\text{Min} \quad \sum_{o \in V} \sum_{(i,j) \in A} c_{ij} f_{ij}^{o} \tag{3.16a}$$

$$\text{s.t.} \quad \sum_{\{i,j\} \in E} x_{ij} = |V| - 1 \tag{3.16b}$$

$$\sum_{i:(i,o) \in A} f_{io}^{o} = 0 \qquad \forall\, o \in V \tag{3.16c}$$

$$\sum_{i:(i,j) \in A} f_{ij}^{o} - \sum_{k:(j,k) \in A} f_{jk}^{o} = w^{oj} \qquad \forall\, o \in V \quad \forall\, j \in V \setminus \{o\} \tag{3.16d}$$

$$\sum_{k:(o,k) \in A} f_{ok}^{o} = \sum_{d:(o,d) \in R} w^{od} \qquad \forall\, o \in V \tag{3.16e}$$

$$f_{ij}^{o} + f_{ji}^{o} \leq \left(\sum_{d:(o,d) \in R} w^{od} \right) x_{ij} \qquad \forall\, o \in V \quad \forall\, \{i,j\} \in E \tag{3.16f}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, \{i,j\} \in E \tag{3.16g}$$

$$f_{ij}^{o} \geq 0 \qquad \forall\, o \in V \quad \forall\, (i,j) \in A \tag{3.16h}$$

Now, in addition to the design variables x_{ij}, we use the $f^o_{ij}(=\sum_d y^{od}_{ij})$ variables that represent the sum of flows, with origin at o that traverse the arc (i,j).

We should expect, however, weaker linear relaxation bounds from such aggregated formulation, which translates into bigger search trees when an enumeration-based method is applied to this formulation (thus limiting its effective use to solving medium size instances).

3.3.3 The Rooted tree formulation

Traditional linear integer programming formulations for the OCSTP are based on the y^r_{ij} *path* variables or the f^o_{ij} *flow* variables and, thus, require a large amount of working memory. In contrast, the new linear formulation presented below makes use of $\mathcal{O}(n^2)$ variables and $\mathcal{O}(n^3)$ constraints. To the best of our knowledge, this is the most compact linear formulation for the OCSTP.

Rooted tree formulation:

$$\text{Min} \quad \sum_{r\in R} w^r d^r \tag{3.17a}$$

$$\text{s.t.} \quad \sum_{i:(i,j)\in A} x_{ij} \;=\; 1-\delta^o_j \qquad \forall\, j\in V \tag{3.17b}$$

$$x_{ij} \le p_{ij} \qquad \forall\,(i,j)\in A \tag{3.17c}$$

$$p_{ij}+p_{ji} \le 1 \qquad \forall\,(i,j)\in A \tag{3.17d}$$

$$p_{ij}+x_{jk} \le 1+p_{ik} \qquad \forall\,\{i,j,k\}\in \tbinom{V}{3} \tag{3.17e}$$

$$p_{ik}+x_{jk} \le 1+p_{ij} \qquad \forall\,\{i,j,k\}\in \tbinom{V}{3} \tag{3.17f}$$

$$d^{ij} \ge d^{ik}+c_{kj}-M(2-x_{kj}-p_{ik}) \qquad \forall\,\{i,j,k\}\in \tbinom{V}{3} \tag{3.17g}$$

$$d^{ij} \ge d^{ik}+c_{kj}-M(1-x_{kj}+p_{ij}+p_{ji}) \qquad \forall\,\{i,j,k\}\in \tbinom{V}{3} \tag{3.17h}$$

$$x_{ij}\in\{0,1\} \qquad \forall\,(i,j)\in A \tag{3.17i}$$

$$p_{ij}\in\{0,1\} \qquad \forall\,(i,j)\in A \tag{3.17j}$$

$$d^{ij} \;\ge\; c_{ij}(x_{ij}+x_{ji}) \qquad \forall\,\{i,j\}\in E \tag{3.17k}$$

where M is an upper bound on the length of the longest path of G, $\delta^o_j=1$ if $j=o$ (and 0 otherwise) and $\{i,j,k\}\in\binom{V}{3} \;\Leftrightarrow\; i,j,k\in V$ and $i\neq j\neq k\neq i$.

In addition to the binary design variables x_{ij}, this formulation uses the *precedence* auxiliary variables p_{ij}, that indicate whether or not there is a directed path from i to j and can be interpreted as a partial order on the vertices of V. The correctness of this formulation, however, is not obvious, so below we explain how the (3.17b)-(3.17f) constraints work coordinately to model directed spanning trees.

Constraints (3.17b) control the indegree of every vertex in the final solution and imply that only $n-1$ x variables can be active simultaneously. Thus, if we are able to prevent directed or undirected cycles in the x variables, the only compatible topology will be a directed tree rooted at vertex o. Indeed, constraints (3.17b) imply that no undirected cycle is possible, since every undirected cycle has, at least, one vertex with indegree strictly greater than 1.

To avoid directed cycles, we need the auxiliary variables p, which determine if there is a directed path between every pair of vertices. They are related to x by (3.17c) as well as by the directed path constraints (3.17e), ensuring that, when there is a directed path of active variables x connecting i and j, the variable p_{ij} is also activated. Now, we can prevent all directed cycles on the x variables using the (3.17d) constraints (by the contrapositive of the previous argument).

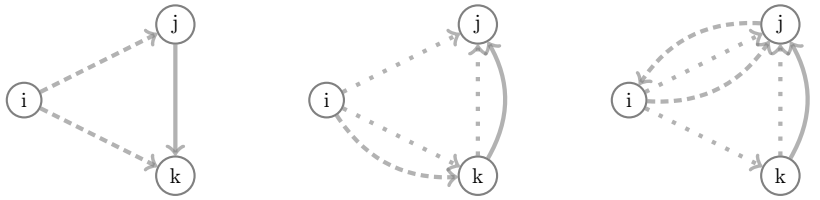


Figure 3.1: Graphic representation of (3.17e) and (3.17f) (left), (3.17g) (center) and (3.17h) (right).

Finally, to avoid false activation of the p variables (i.e. having $p_{ij} = 1$ when there is no directed path from i to j) we introduce the set of constraints (3.17f). They prevent the incorrect activation of a p_{ij} variable because we can always find a common ancestor k of both, i and j, by following the corresponding directed paths backwards, and then find a violated constraint of type (3.17d). Note that it is not necessary to avoid the false activation of the x variables because only a fixed number of them, $n-1$, can be active simultaneously.

Once the rooted tree topology is correctly modeled by the x_{ij} and p_{ij} variables, and taking into account that the objective (3.17a) is to minimize a non-negative sum of d^{ij} variables, we only need to lower bound the distances d^{ij} between all pairs of vertices so they take their correct values.

With (3.17g) we follow directed paths of x_{ij} variables forward (as in (3.17e)) to increase the distance between a pair of *comparable* vertices (when there is a directed path between them) and with (3.17h) we follow directed paths of x_{ij} variables backwards (as in (3.17f)) in order to increase the distance between a pair of *incomparable* vertices (when there is not a directed path between them).

Several improvements can be applied to the above formulation. First, we can relax the integrality constraints (3.17j) since (3.17b)–(3.17f) and (3.17i) are enough to guarantee the integrality of the p_{ij} variables. Indeed, when there is a directed path from i to j, the combination of (3.17c), (3.17e) and (3.17i) raises the value of p_{ij} to 1. Conversely, if the x_{ij} do not form a directed path from i to j, the combination of (3.17d), (3.17f) and (3.17i) imposes that $p_{ij} = 0$.

Going further, we can substitute families (3.17e) and (3.17f) by a single new family of constraints of the form:

$$p_{ij} + p_{jk} + x_{kj} \leq 1 + p_{ik} \tag{3.18}$$

Indeed, constraints (3.17e) ensure that if $p_{ij} = 1$ and $x_{jk} = 1$, then $p_{ik} = 1$. With (3.18) we ensure that if $p_{ij} = 1$ and $p_{jk} = 1$ then $p_{ik} = 1$, which is stronger, and can be particularized using (3.17c). Likewise, constraints (3.17f) ensure that if $p_{ij} = 1$ and $x_{kj} = 1$, then $p_{ik} = 1$. Again, with (3.18) we ensure the same condition, this time without the help of (3.17c).

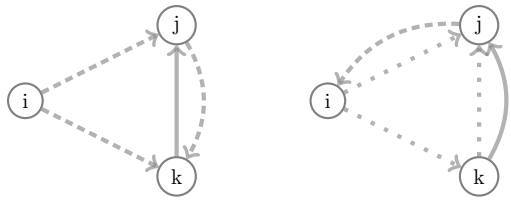


Figure 3.2: Graphic representation of (3.18) (left) and (3.19) (right).

Families (3.17g) and (3.17h) can also be replaced by a new family of tighter constraints of the form:

$$d^{ij} \geq d^{ik} + c_{kj} - M(1 - x_{kj} - p_{ji}) \tag{3.19}$$

In this case, (3.17g) ensure that if $p_{ik} = 1$ and $x_{kj} = 1$, then $d^{ij} \geq d^{ik} + c_{kj}$. With (3.19) we ensure that if $x_{kj} = 1$ and $p_{ji} = 0$ then $d^{ij} \geq d^{ik} + c_{kj}$, which is more general, and can be particularized using that if $p_{ik} = 1$ and $x_{kj} = 1$, then $p_{ji} = 0$. Similarly, (3.17h) ensure that if $p_{ij} = p_{ji} = 0$ and $x_{kj} = 1$, then $d^{ij} \geq d^{ik} + c_{kj}$ whereas (3.19) disregards the value uf p_{ij}, providing a more general and stronger cut.

Finally, we can substitute (3.17k) by the stronger:

$$d^{ij} \geq c_{ij}(x_{ij} + x_{ji}) + c^2_{ij}(1 - x_{ij} - x_{ji}) \tag{3.20}$$

where c^2_{ij} is the length of the second-shortest-path of G connecting i and j.

Applying all these improvements to (3.21), we obtain the following linear formulation for the OCSTP:

Rooted tree formulation 2:

$$\text{Min} \quad \sum_{r \in R} w^r d^r \tag{3.21a}$$

$$\text{s.t.} \quad \sum_{i:(i,j)\in A} x_{ij} \;=\; 1 - \delta^1_j \qquad \forall\, j \in V \tag{3.21b}$$

$$x_{ij} \leq p_{ij} \qquad \forall\, (i,j) \in A \tag{3.21c}$$

$$p_{ij} + p_{ji} \leq 1 \qquad \forall\, (i,j) \in A \tag{3.21d}$$

$$p_{ij} + p_{jk} + x_{kj} \leq 1 + p_{ik} \qquad \forall\, \{i,j,k\} \in \tbinom{V}{3} \tag{3.21e}$$

$$d^{ij} \geq d^{ik} + c_{kj} - M(1 - x_{kj} - p_{ji}) \qquad \forall\, \{i,j,k\} \in \tbinom{V}{3} \tag{3.21f}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in A \tag{3.21g}$$

$$p_{ij} \;\geq\; 0 \qquad \forall\, (i,j) \in A \tag{3.21h}$$

$$d^{ij} \;\geq\; c_{ij}(x_{ij} + x_{ji}) + c^2_{ij}(1 - x_{ij} - x_{ji}) \qquad \forall\, \{i,j\} \in E \tag{3.21i}$$

The most obvious drawback of formulation (3.21) is the use of big values of M, so weak lower bounds are obtained by its linear relaxation. This latter fact

makes (3.21) impractical for solving general instances of the OCSTP. However, the formulation remains interesting from a theoretical point of view, because it is closely related to a well-known formulation of the Linear Ordering Problem (if we substitute $p_{ij} + p_{ji} \leq 1$ by $p_{ij} + p_{ji} = 1$ above we obtain a valid formulation for the LOP (Martí and Reinelt, 2011)), thus, advances in the study of the LOP polytope may lead to the discovery of valid inequalities for (3.21).

3.3.4 Computational comparison of linear formulations

We can obtain optimal solutions to small OCSTP instances using modern MILP software, such as CPLEX or Xpress, as a black-box. However, in order to solve larger instances it is necessary to develop more sophisticated algorithms and stronger formulations. In Chapter 5 we propose a Benders reformulation of the path-based formulation described above, as well as a Branch & Cut algorithm which is able to solve medium size instances. Following the advice of Magnanti and Wong (1981), we choose the path-based formulation as a base for our Benders reformulation because it has a suitable structure for decomposition methods and provides the best linear relaxation bounds. In this section we compare the linear relaxation bounds of the linear formulations described above.

We used 20 OCSTP instances from Contreras et al. (2010a), ranging from 10 to 50 vertices each, as a test bed for our computational experiments (see Appendix B for a detailed description). The linear relaxation of the path-based formulation (3.15), the flow-based formulation (3.16), and the two rooted-tree formulations (3.17) and (3.21), have been implemented and CPLEX has been used as a black-box solver. Then the optimal solution has been compared with the best known solutions for these instances (see Section 5.3) to obtain the optimality gaps ($gap = \frac{z_{upp} - z_{lwr}}{z_{upp}}$) summarized in Table 3.2.

From the table we can clearly see that the path-based formulation provides better linear relaxation bounds whereas the other formulations are roughly equivalent. We have developed, however, several families of valid inequalities that might improve the flow-based formulation.

Let m_{ij} be the matrix of minimum cuts separating i and j in G with respect to the capacity vector w^{ij}. The m_{ij} matrix can be computed from the corresponding MCT in polynomial time as a pre-processing step. By definition, if the design variable x_{ij} appears in the optimal solution T^*, it is evident that $T^* \setminus \{\{i, j\}\}$

|V|	P-LP	F-LP	R1-LP	R2-LP
10	0.25%	11.29%	11.37%	9.44%
20	2.16%	16.50%	16.52%	9.68%
30	2.41%	17.66%	17.68%	13.45%
40	4.89%	19.92%	19.92%	16.41%
50	6.84%	21.57%	21.57%	19.26%

Table 3.2: Average optimality gaps for the linear relaxation of (3.15), (3.16), (3.17) and (3.21) applied to instances from Contreras et al. (2010a).

has two connected components that must use the edge $\{i, j\}$ to satisfy their communication requirements. Therefore, the amount of flow that traverses $\{i, j\}$ must be at least m_{ij}, since this is a lower bound on all the cuts that separate i and j.

$$m_{ij}x_{ij} \leq \sum_{o \in V \setminus \{j\}} f^o_{ij} + \sum_{o \in V \setminus \{i\}} f^o_{ji} \qquad \{i, j\} \in E. \tag{3.22}$$

We also introduce a new set of y^o_{ij} auxiliary design variables representing the discrete version of the f^o_{ij} flow variables (where $y^o_{ij} = 1$ if $f^o_{ij} > 0$). Thus, for a given vertex o, the set of y^o_{ij} variables represent a directed subtree rooted at that vertex. We go, however, a little bit further and impose that these directed trees should span the whole graph (even if some communication requirements are 0 for a given (o, d) pair).

We need the following constraints to correctly model the directed spanning trees represented by y^o_{ij}:

$$y^o_{ij} + y^o_{ji} = x_{ij} \qquad \forall\, \{i, j\} \in E, \forall\, o \in V, \tag{3.23a}$$

$$f^o_{ij} \leq \left(\sum_{d:(o,d) \in R} w^{od} \right) y^o_{ij} \qquad \forall\, (i, j) \in A, \forall\, o \in V, \tag{3.23b}$$

$$y^o_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in A, \forall\, o \in V, \tag{3.23c}$$

but we can also add these other simple families to further constraint the range of values that the y^o_{ij} variables can adopt:

$$\sum_{i:(i,j)\in A} y^o_{ij} = 1 \qquad \forall\, o,j \in V,\ o \neq j, \tag{3.24a}$$

$$\sum_{i:(i,o)\in A} y^o_{io} = 0 \qquad \forall\, o \in V, \tag{3.24b}$$

$$\sum_{j:(o,j)\in A} y^o_{oj} \geq 1 \qquad \forall\, o \in V. \tag{3.24c}$$

Now, we can use these directed trees to restate constraints (3.22) as:

$$\lambda^o_{ij} y^o_{ij} + \lambda^o_{ji} y^o_{ji} \leq \sum_{u\in V\setminus\{j\}} f^u_{ij} + \sum_{u\in V\setminus\{i\}} f^u_{ji} \qquad \{i,j\}\in E,\ \forall o \in V, \tag{3.25}$$

where $\lambda^o_{ij} = \max\{m_{oj}, m_{ij}\}$ and $\lambda^o_{ji} = \max\{m_{oi}, m_{ji}\}$.

We can also lower-bound the flow that must traverse each arc:

$$w^{oj} y^o_{ij} \leq f^o_{ij} \qquad \forall \{i,j\}\in E, \quad \forall o \in V. \tag{3.26}$$

and take into account nearby arcs to improve further these lower bounds:

$$(w^{ok}+w^{ol})y^o_{kl} \leq \sum_{i\in V\setminus\{k\}} f^o_{ik} - \sum_{j\in V\setminus\{k,l\}} f^o_{kj} \qquad \forall\, o,k,l \in \tbinom{V}{3}, \tag{3.27a}$$

$$(w^{ok}+w^{ol}+w^{oh})(y^o_{kl}+y^o_{lh}-1) \leq \sum_{i\in V\setminus\{k\}} f^o_{ik} - \sum_{j\in V\setminus\{k,l,h\}} f^o_{kj} \quad \forall\, o,k,l,h \in \tbinom{V}{4}, \tag{3.27b}$$

$$(w^{ok}+w^{ol}+w^{og})(y^o_{kl}+y^o_{kg}-1) \leq \sum_{i\in V\setminus\{k\}} f^o_{ik} - \sum_{j\in V\setminus\{k,l,g\}} f^o_{kj} \quad \forall\, o,k,l,g \in \tbinom{V}{4}, \tag{3.27c}$$

where $\binom{V}{\kappa}$ is the set of all permutations of κ different elements of V.

Note, however, that (3.27b) and (3.27c) are $\mathcal{O}(n^4)$ families of constraints, and, therefore, they must be separated dynamically if this formulation is used on medium-to-large size instances.

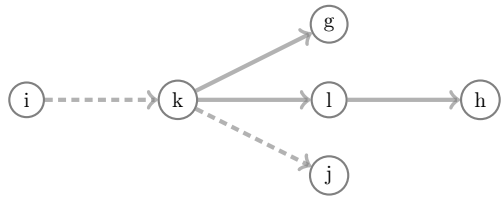


Figure 3.3: Graphic representation of (3.27a), (3.27b) and (3.27c).

All of these families of valid inequalities improve the flow-based formulation by taking into account the sense of the flow and not only its volume. However, the linear relaxation of the flow-based formulation still provides weaker lower bounds than the linear relaxation of the path-based formulation. These results are summarized in Table 3.3, where the **F2-LP** column contains the results obtained by the flow-based formulation (3.16) extended with the (3.22)-(3.27a) valid inequalities, whereas **F3-LP** also includes the (3.27b) and (3.27c) $\mathcal{O}(n^4)$ families.

|V|	P-LP	F1-LP	F2-LP	F3-LP	R1-LP	R2-LP
10	0.25%	11.29%	1.13%	0.79%	11.37%	9.44%
20	2.16%	16.50%	7.82%	6.27%	16.52%	9.68%
30	2.41%	17.66%	8.07%	6.30%	17.68%	13.45%
40	4.89%	19.92%	11.01%	9.63%	19.92%	16.41%
50	6.84%	21.57%	13.01%	10.93%	21.57%	19.26%

Table 3.3: Average optimality gaps for the linear relaxation of (3.15), (3.16), (3.16) & (3.22)-(3.27a), (3.16) & (3.22)-(3.27c), (3.17) and (3.21) applied to instances from Contreras et al. (2010a).

Chapter 4

Heuristic algorithms for the OCSTP

The NP-Hard nature of the OCSTP and, more remarkably, the actual difficulty of finding optimal solutions to OCSTP instances in practice, justify the need of developing good heuristic algorithms that are able to provide high quality solutions when the size of the instances exceeds the capacity of the exact algorithms. In addition, near-to-optimum solutions can provide useful information to exact algorithms, such as tight lower bounds that allow pruning significant parts of the solution space in an enumerative framework, reducing the computational effort needed to make the algorithm converge to the optimum solution.

In Section 4.1 we propose a novel Divide & Conquer constructive heuristic specially designed for the OCSTP.

Section 4.2 is devoted to improvement heuristic algorithms. We start with a review of the 1-edge-exchange neighborhood and the associated Ahuja-Murty Local Search. Next we show that the efficient technique proposed by Ahuja and Murty can be extended to a significant subset of the 2-edge-exchange neighborhood without altering the asymptotical cost of the algorithm. Finally, we introduce a new family of spanning tree neighborhoods associated with the Dandelion Code and a suitable exploration strategy for the Partially Ordered Neighborhood Structure defined by the Dandelion Neighborhoods.

4.1 Constructive heuristics

Since all spanning trees are feasible solutions of the OCSTP, any algorithm returning an element of $\mathcal{T}_n$ may be considered a constructive heuristic for the OCSTP. In fact, finding spanning trees with a low communication cost is not difficult for most instances of the OCSTP and striving for trees that minimize length (Minimum Spanning Trees) or capacity (Min-Cut Trees) may be enough for many applications. It is, however, fundamental to understand that most of these simple heuristics may fail to find an optimal solution for a wide range of instances and therefore it is important to design constructive heuristics specifically tailored for the OCSTP.

The following instance:

dist	1	2	3	4	5
1	0	1	2	2	2
2	1	0	1	2	2
3	2	1	0	1	2
4	2	2	1	0	1
5	2	2	2	1	0

reqs	1	2	3	4	5
1	0	1	20	20	1
2	0	0	1	1	20
3	0	0	0	1	20
4	0	0	0	0	1
5	0	0	0	0	0

illustrates how different optimal solutions to the MST and the OCSTP can be for a given distance matrix. As can be seen in Figure 4.1, the optimal MST and the OCSTP do not share any edge, which does not mean that their communication cost differ greatly. For this particular instance the communication cost of the MST is 210 whereas the communication cost of the OCST is 192.

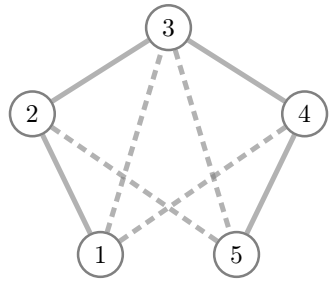


Figure 4.1: OCST (dashed lines) versus MST (bold lines).

In this section we propose a new constructive heuristic specially designed for the OCSTP. Before describing our Divide & Conquer heuristic, however, we are going to review the well-known Ahuja-Murty constructive heuristic, which introduces some basic notions that will clarify our approach.

Both heuristics exploit the same properties of spanning trees but each one does it in a different way. While the Ahuja-Murty constructive heuristic builds a tree iteratively as a single connected component by fixing one edge at a time, our Divide & Conquer heuristic splits the vertex set into n single-vertex subsets and then merges two of them at a time until a single connected component is obtained.

4.1.1 The Ahuja-Murty Constructive Heuristic

The constructive heuristic presented in Ahuja and Murty (1987) builds a tree one edge at a time, always maintaining a single connected component. At each iteration, the heuristic chooses a vertex not yet in the tree and connects it to the current subtree using the locally optimal edge for this connection.

The algorithm starts by selecting an arbitrary vertex i, marking it as visited ($S = \{i\}$), and by defining the initial partial solution as $T = \{\varnothing\}$. Once initialized, the algorithm performs $n - 1$ iterations, selecting a new edge $\{i, j\}$, with $i \in S$ and $j \notin S$, and updating both, the set of visited vertices ($S = S \cup \{j\}$) and the current partial solution ($T = T \cup \{i, j\}$).

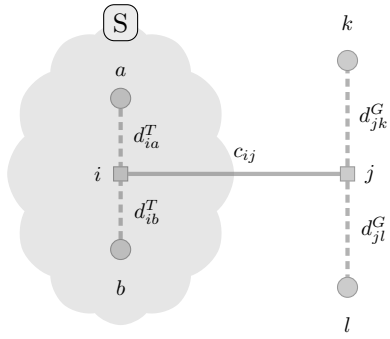


Figure 4.2: T grows as a single connected component, one edge at each iteration.

The criterion suggested by Ahuja-Murty is to select the edge $\{i, j\}$ that minimizes:

$$\xi_i + c_{ij}\Omega + \tilde{\xi}_j \tag{4.1}$$

where Ω is the total amount of communication that must traverse the cut $\{S, V \setminus S\}$, ξ_i is the cost of moving Ω communication to the exiting vertex i and $\tilde{\xi}_j$ is an heuristic estimation of the same value for vertex j (since ξ_j can not be computed exactly until the rest of the tree has been built). These quantities can be computed efficiently in terms of ω_i's and ω_j's.

$$\Omega = \sum_{i \in S} \omega_i \tag{4.2}$$

with

$$\begin{aligned} \omega_i &= \sum_{j \notin S} w^{ij} && \forall\, i \in S \\ \omega_j &= \sum_{i \in S} w^{ij} && \forall\, j \notin S, \end{aligned} \tag{4.3}$$

and

$$\begin{aligned} \xi_i &= \sum_{k \in S} \omega_k d^T_{ik} && \forall\, i \in S \\ \tilde{\xi}_j &= \sum_{k \notin S} \omega_k d^G_{kj} && \forall\, j \notin S, \end{aligned} \tag{4.4}$$

where we can compute the exact value of ξ_i since we already know d^T_{ik}, the length of the unique path of T connecting vertices of S, but we can only compute the approximate value of $\tilde{\xi}_j$ using d^G_{kj}, the length of the shortest path of G connecting vertices of $V \setminus S$.

Note that when vertex j enters S, we are making the assumption that $\tilde{\xi}_j$ is a good approximation of the communication cost of moving Ω to j before crossing the edge $\{i, j\}$ and spend an additional ξ_i communication cost to arrive to its destination. When the set S contains few vertices, this approximation may be misleading, since many vertices are on $V \setminus S$ and the final distances over T will greatly differ from the shortest path distances used to compute $\tilde{\xi}_j$.

However, Ahuja-Murty constructive heuristic still remains useful for the wide

family of instances where the lengths of shortest paths of G constitute a good indicator of the lengths of the (unique) paths connecting the vertices of T. Moreover, the computational effort required to build this feasible solution ($\mathcal{O}(n^3)$ operations) is acceptable, specially for those cases where no improvement heuristic will be applied a posteriori.

4.1.2 The Divide & Conquer Heuristic

Below we present a new constructive heuristic for the OCST. It is based on the Divide & Conquer paradigm and can be defined recursively as follows: break the vertex set V into two subsets (V_1, V_2), apply the Divide & Conquer heuristic to both subsets to obtain two subtrees (T_1, T_2) spanning them and finally select a locally optimal edge $\{i, j\}$, with $i \in T_1$ and $j \in T2$, and return $T = T_1 \cup \{i, j\} \cup T_2$.

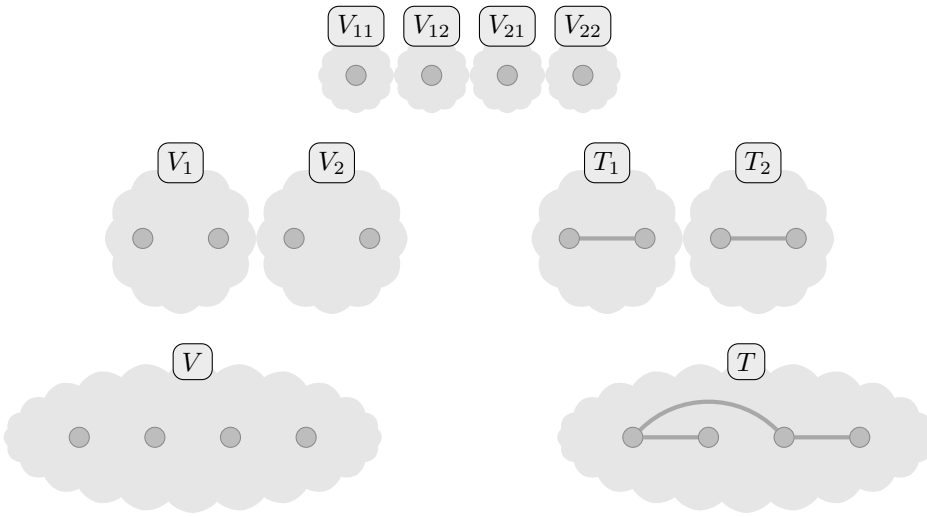


Figure 4.3: Recursive Divide & Conquer heuristic.

Both, the criterion used to split V and the criterion used to merge T_1 and T_2, influence the behavior of this heuristic. In particular, we must note that the splitting criterion does not fully determine the shape of the final T, since splitting V_k determines a cut on the vertices of V_k but not how T_k will be connected to the rest of the graph.

There is an alternative bottom up version of this Divide & Conquer heuristic, where we initialize the algorithm with n nodes, each one containing a single vertex of G, and then apply $n-1$ merging operations to obtain T. In this alternative version of the algorithm, the role of the splitting criterion is played by the criterion used to select the nodes that are going to be merged in each iteration. Otherwise, both alternatives can be considered equivalent. Through this new interpretation, we can clearly see that the Divide & Conquer heuristic is analogue to the Kruskal's MST algorithm while the Ahuja-Murty constructive heuristic is analogue to Prim's MST algorithm.

Since the internal structure of T_1 and T_2 is already known at the moment of selecting $\{i, j\}$, we can compute the exact communication cost of routing, through $\{i, j\}$, the communications between vertices of T_1 and vertices of T_2:

$$\xi_i + c_{ij}\Omega + \xi_j \tag{4.5}$$

Note, however, that the values of ξ_i do not take into account the communication cost of vertices outside $V_1 \cup V_2$, since they are now defined as:

$$\begin{aligned} \xi_i &= \sum_{k \in V_1} \omega_k d_{ik}^{T_1} && \forall\, i \in V_1 \\ \xi_j &= \sum_{k \in V_2} \omega_k d_{kj}^{T_2} && \forall\, j \in V_2, \end{aligned} \tag{4.6}$$

with

$$\begin{aligned} \omega_i &= \sum_{j \in V_2} w^{ij} && \forall\, i \in V_1 \\ \omega_j &= \sum_{i \in V_1} w^{ij} && \forall\, j \notin V_2, \end{aligned} \tag{4.7}$$

and

$$\Omega = \sum_{i \in V_1} \omega_i. \tag{4.8}$$

In contrast to the Ahuja-Murty algorithm, the Divide & Conquer heuristic does not try to approximate the communication cost for sending information to vertices outside $V_1 \cup V_2$. It rather disregards these quantities, finding a (locally) optimum merge of T_1 and T_2 and relying on the capacity of future merges to

efficiently connect $T_1 \cup T_2$ with the rest of the graph. Therefore, it is fundamental to ensure that these communication costs can be safely ignored through the use of a suitable splitting criterion. One natural way to do so is to split V into V_1 and V_2 using a minimum cut of V with respect to the capacity vector defined by the communication requirements w^r. Alternatively, if we are using the bottom-up version of the algorithm, we can select to merge T_i and T_j if, among all possible $\{i, j\}$ pairs, they maximize their information exchange:

$$\sum_{o \in T_i} \sum_{d \in T_j} w^{od}. \tag{4.9}$$

This latter version of the algorithm can be implemented easily using a simple union-find data structure that is updated in time $\mathcal{O}(n)$ after each merge operation. On the other hand, each of the $n-1$ merge operations can be performed in time $\mathcal{O}(|V_1| \cdot |V_2|)$ but it is well known that

$$\binom{n}{2} = \binom{k}{2} + k(n-k) + \binom{n-k}{2} \qquad \forall 0 \leq k \leq n, \tag{4.10}$$

which means that, as long as the quadratic merging time dominates the time required to select the subtrees to be merged, the total time required to perform the whole bottom-up Divide & Conquer constructive heuristic is $\mathcal{O}(n^2)$. Note that this is faster than the Ahuja-Murty constructive heuristic.

Finally, after each merge operation, a local search procedure (such as the Ahuja-Murty Local Search described in Section 4.2.1) can be applied to the resulting subtree to correct previous *mistakes* before they can influence future merges. The objective of these intermediate self-correcting steps is to increase the chances of escaping from local optimum trees that will otherwise be obtained if the improving heuristic is applied only at the end of the Divide & Conquer heuristic.

4.2 Improvement heuristics

The large size of the OCSTP solution space represents a real challenge for any improvement heuristic. Even the relatively small 1-edge-exchange neighborhood of a single tree has $\mathcal{O}(n^3)$ elements, consequently, most deterministic improvement heuristics for the OCSTP require at least $\mathcal{O}(n^3)$ operations per improvement. In this section we present an extension of the well-known Ahuja-Murty Local Search that meets this $\mathcal{O}(n^3)$ limit while exploring part of the 2-edge-exchange neighborhood. We will also propose a completely new family of spanning tree neighborhoods (each of them containing between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^{n-2})$ elements) along with a suitable exploration strategy that exploits their structural features.

4.2.1 The Ahuja-Murty Local Search

Evaluating the communication cost of a single tree T requires time $\mathcal{O}(n^2)$, in general, since it is necessary to compute $|R|$ distances over T, and R induces a connected subgraph on G (i.e. $|R| \geq n-1$ and each distance may require up to n operations to be computed).

The number of trees that differ in just 1 edge from a given tree T (the 1-edge-exchange neighborhood of T) depends on the structure of T. A star has a 1-exchange-neighborhood of size $\mathcal{O}(n^2)$ whereas a path has a 1-exchange-neighborhood of size $\mathcal{O}(n^3)$. Both are extreme cases but the expected size of such neighborhoods is $\mathcal{O}(n^3)$ in general.

Taking these two facts into account, it is evident that a naïve algorithm may require $\mathcal{O}(n^5)$ operations (in the worst case) to evaluate the elements of the 1-edge-exchange neighborhood of an arbitrary tree. However, Ahuja and Murty (1987) presented a really clever algorithm that exploits the natural structure of the problem to explore such neighborhoods in just $\mathcal{O}(n^3)$ operations, evaluating each tree in constant amortized time.

The Ahuja-Murty local search starts by pre-computing d_{ij}, the distance matrix of T and $z = \sum_r w^r d_{o^r d^r}$, the communication cost of T, in $\mathcal{O}(n^2)$ total time. After this global pre-processing step, the algorithm iterates over the $n-1$ edges of T and for each of them it applies the following process: Let e be the selected edge of T and $(S, \bar{S})$ the cut associated with $T \setminus \{e\}$. For every $i \in V$, the amount of communication that vertex i needs to send to the vertices of the other side of

the cut is computed:

$$\omega_i^e = \begin{cases} \sum_{j \in \bar{S}} w^{ij} & \text{if } i \in S \\ \sum_{j \in S} w^{ij} & \text{if } i \in \bar{S}. \end{cases} \tag{4.11}$$

Then the total amount of communication that must traverse the cut is:

$$\Omega^e = \sum_{i \in S} \omega_i^e, \tag{4.12}$$

and the total cost of moving ω_j^e to vertex i for all j in the i's side of the cut:

$$\xi_i^e = \begin{cases} \sum_{j \in S} \omega_j^e d_{ij} & \text{if } i \in S \\ \sum_{j \in \bar{S}} \omega_j^e d_{ij} & \text{if } i \in \bar{S}. \end{cases} \tag{4.13}$$

Determining the elements of S and $\bar{S}$ and computing all these associated values requires $\mathcal{O}(n^2)$ operations and must be considered a pre-processing step associated with edge $e = \{e_1, e_2\}$. Now for every $i \in S$ and for every $j \in \bar{S}$ we can evaluate the communication cost of $(T \setminus \{e\}) \cup \{\{i, j\}\}$ in constant time as:

$$z - \left(\xi_{e_1}^e + \Omega^e \cdot d_e + \xi_{e_2}^e\right) + \left(\xi_i^e + \Omega^e \cdot d_{ij} + \xi_j^e\right). \tag{4.14}$$

Since the global pre-processing phase requires $\mathcal{O}(n^2)$ operations, each of the $n - 1$ edge-pre-processing phases requires $\mathcal{O}(n^2)$ steps, and evaluating each of the $\mathcal{O}(n^3)$ neighbors of T requires $\mathcal{O}(1)$ operations, we can explore the whole 1-edge-exchange neighborhood of T in $\mathcal{O}(n^3)$ time.

The above exploration strategy can be used to implement a simple best-improvement local search that examines the whole 1-edge-exchange neighborhood of T and selects the best neighbor iteratively until a local minimum is reached. Alternatively, the exploration of each neighborhood can be stopped as soon as it finds a tree with smaller communication cost. For such first-improvement local search, suitable strategies (such as selecting the edges by decreasing order of length or using a FIFO queue to ensure that the most recently added edge is examined in last term) may reduce the amount of time needed to find a tree with smaller communication cost. Nevertheless, in order to verify that a given tree is,

in fact, a local optimum, it may be necessary to perform a final $\mathcal{O}(n^3)$ search.

The Ahuja-Murty local search (AMLS) remains as one of the most effective local search algorithms for the OCSTP, combining simplicity, flexibility and good performance in practice. As a consequence, many heuristic algorithms are based on the AMLS and several improvements to the basic scheme had been proposed in the literature.

Wolf and Merz (2010) describe an efficient exploration of the 1-edge-exchange neighborhood, for the special case where all communication requirements are equal, with the same worst case cost as the AMLS, but which improves its expected case cost to $\mathcal{O}(n^2 \log n)$. In Fischer and Merz (2007) the AMLS is used to boost a memetic algorithm for the OCSTP. Fischer (2007) proposes a variant of the AMLS that only computes the ξ_i^e values for a subset of the vertices of S and $\bar{S}$, reducing the amount candidate edges to be included in the tree in order to increase the overall speed of the search.

Extending the AMLS to the 2-edge-exchange neighborhood

The AMLS 1-edge-exchange can be extended to a restricted (but nonetheless huge) section of the 2-edge-exchange neighborhood at the expenses of pre-computing and storing some additional auxiliary data.

We start as before by pre-computing d_{ij}, the distances over T between each pair of vertices, and z, the communication cost z of T. Then, for each edge e of T, we pre-compute and store the cut $\{S^e, \bar{S}^e\}$ associated with $T \setminus \{e\}$. This operation requires time $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2)$ memory. Finally, we pre-compute and store ω_i^e, Ω^e and ξ_i^e as defined before. This operation uses time $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ memory.

Now, if e and f are two edges of T, with $\{A, B \cup C\}$ being the cut associated with $T \setminus \{e\}$ and $\{A \cup B, C\}$ the cut associated with $T \setminus \{f\}$, we can define $\Omega^{ef} = \Omega^{fe}$ as the amount of communication between A and C:

$$\Omega^{ef} = \sum_{a \in A} \sum_{c \in C} w^{ac}. \tag{4.15}$$

Note that Ω^{ef} accounts for the amount of communication that traverses both, the edge e and the edge f. In fact, this definition is perfectly compatible with $\Omega^{ee} = \Omega^e$.

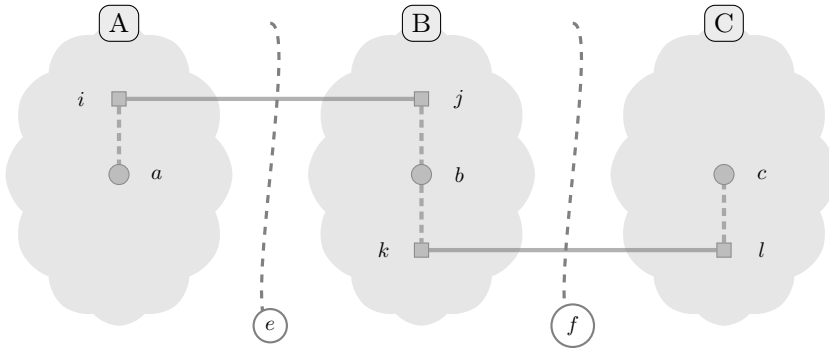


Figure 4.4: Removing 2 edges breaks T into A, B & C.

Note also that we can compute each of the $\mathcal{O}(n^2)$ values Ω^{ef} in linear time using the following equivalent expressions:

$$\Omega^{ef} = \sum_{c \in C} \omega_c^e. \tag{4.16}$$

Therefore, all the auxiliary data can be pre-computed in time $\mathcal{O}(n^3)$ and stored using $\mathcal{O}(n^2)$ memory.

Now, if we want to explore the 2-edge-exchange neighborhood of T we must examine all possible trees formed by removing two edges from T and then reconnecting again the corresponding A, B and C subtrees. There are two general cases to consider: whether the subtree B remains between subtrees A and C or not. For the first case (where B is used to connect A and C), we can derive a closed expression that allows us to compute the communication cost of a tree of that restricted 2-edge-exchange neighborhood in constant time from the auxiliary data described above. Thus, we can extend the AMLS technique to a huge subset of the 2-edge-exchange neighborhood with little additional effort. Note that such restricted neighborhoods are potentially enormous, containing between $\mathcal{O}(n^4)$ and $\mathcal{O}(n^5)$ elements, but any subset of them of size $\mathcal{O}(n^3)$ can be explored for free (from an asymptotical perspective) when we perform a classic AMLS.

Let $\tilde{T} = (T \setminus \{\{i,j\},\{k,l\}\}) \cup \{\{u,v\}\{x,y\}\}$ be the new tree, (with $e = \{i,j\}$ and $f = \{k,l\}$) and let a, b and c be generic vertices of their respective subtrees.

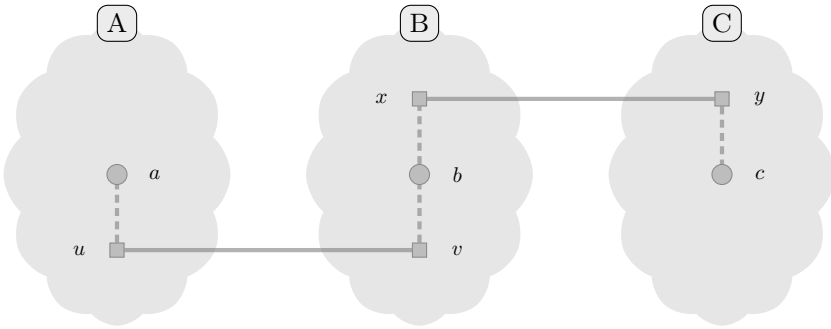


Figure 4.5: General case of the restricted 2-exchange neighborhood.

Then we can derive a closed-form expression for $\Delta = \tilde{z} - z$, the difference between the communication cost of T and the communication cost of $\tilde{T}$:

$$\Delta = \sum_{a\in A}\sum_{b\in B} w^{ab}(\tilde{d}_{ab} - d_{ab}) + \sum_{a\in A}\sum_{c\in C} w^{ac}(\tilde{d}_{ac} - d_{ac}) + \sum_{b\in B}\sum_{c\in C} w^{bc}(\tilde{d}_{bc} - d_{bc}).$$

Writing $\tilde{d}$ in terms of d and c and rearranging terms::

$$\begin{aligned}
&= \sum_{a\in A}\sum_{b\in B} w^{ab}(d_{au} + c_{uv} + d_{vb} - d_{ai} - c_{ij} - d_{jb}) \\
&+ \sum_{a\in A}\sum_{c\in C} w^{ac}(d_{au}+c_{uv}+d_{vx}+c_{xy}+d_{yc}-d_{ai}-c_{ij}-d_{jk}-c_{kl}-d_{lc}) \\
&+ \sum_{b\in B}\sum_{c\in C} w^{bc}(d_{bx} + c_{xy} + d_{yc} - d_{bk} - c_{kl} - d_{lc}).
\end{aligned}$$

$$\begin{aligned}
&= \sum_{a\in A}\sum_{b\in B} w^{ab}(c_{uv} - c_{ij}) + \sum_{a\in A}\sum_{c\in C} w^{ac}(c_{uv} - c_{ij}) + \sum_{b\in B}\sum_{c\in C} w^{bc}(c_{xy} - c_{kl}) \\
&+ \sum_{a\in A}\sum_{c\in C} w^{ac}(c_{xy} - c_{kl}) + \sum_{a\in A}\sum_{c\in C} w^{ac}(d_{vx} - d_{jk}) + \sum_{a\in A}\sum_{b\in B} w^{ab}(d_{au} - d_{ai}) \\
&+ \sum_{a\in A}\sum_{c\in C} w^{ac}(d_{au} - d_{ai}) + \sum_{b\in B}\sum_{c\in C} w^{bc}(d_{yc} - d_{lc}) + \sum_{a\in A}\sum_{c\in C} w^{ac}(d_{yc} - d_{lc}) \\
&+ \sum_{a\in A}\sum_{b\in B} w^{ab}(d_{vb} - d_{jb}) + \sum_{b\in B}\sum_{c\in C} w^{bc}(d_{bx} - d_{bk}).
\end{aligned}$$

Grouping into pre-computed terms:

$$
\begin{aligned}
&= \Omega^{e}(c_{uv} - c_{ij}) \\
&+\Omega^{f}(c_{xy} - c_{kl}) \\
&+\Omega^{ef}(d_{vx} - d_{jk}) \\
&+\xi_{u}^{e} - \xi_{i}^{e} \\
&+\xi_{y}^{f} - \xi_{l}^{f} \\
&+\sum_{a \in A}\sum_{b \in B} w^{ab}(d_{vb} - d_{jb}) + \sum_{b \in B}\sum_{c \in C} w^{bc}(d_{bx} - d_{bk}).
\end{aligned}
$$

In order to translate the last term into a pre-computed expression we must note that:

$$
\xi_{v}^{e} - \xi_{j}^{e} = \sum_{a \in A}\sum_{b \in B} w^{ab}(d_{vb} - d_{jb}) + \sum_{a \in A}\sum_{c \in C} w^{ac}(d_{kv} - d_{kj})
$$

and:

$$
\xi_{x}^{f} - \xi_{k}^{f} = \sum_{a \in A}\sum_{c \in C} w^{ac}(d_{jx} - d_{jk}) + \sum_{b \in B}\sum_{c \in C} w^{bc}(d_{bx} - d_{bk}).
$$

since the expressions $(d_{kv} - d_{kj})$ and $(d_{jx} - d_{jk})$ do not depend on a or c we can factor them out to obtain:

$$
\begin{aligned}
\Delta &= \Omega^{e}(c_{uv} - c_{ij}) + \Omega^{f}(c_{xy} - c_{kl}) + \Omega^{ef}(d_{vx} - d_{jk}) \\
&+\xi_{u}^{e} - \xi_{i}^{e} + \xi_{y}^{f} - \xi_{l}^{f} \\
&+\xi_{v}^{e} - \xi_{j}^{e} + \xi_{x}^{f} - \xi_{k}^{f} - \Omega^{ef}(d_{kv} - d_{kj} + d_{jx} - d_{jk}),
\end{aligned}
$$

which is a closed-form expression of Δ that can be computed in constant time using only pre-computed terms.

4.2.2 The Dandelion Partially Ordered Neighborhood Structure

In this section we present an extension of the Dandelion code that is able to represent not only spanning trees but also neighborhoods of $\mathcal{T}_n$ of different sizes. These neighborhoods form a Partially Ordered Neighborhood Structure (PONS) that has many interesting structural features. We exploit some of these features by proposing an ad hoc exploration strategy guided by a series of Monte-Carlo trials.

The Dandelion code (see Section 2.1.1, the Appendix A and references therein) provides a compact and unbiased representation of spanning trees of $\mathcal{T}_n$ in the form of words of length $n-2$ in the alphabet $\{1, 2, \ldots, n\}$ (called Cayley Strings and denoted $\mathcal{C}_n$). Amongst all known Cayley codes (bijections between $\mathcal{T}_n$ and $\mathcal{C}_n$), the Dandelion code is characterized for having the smallest locality bound: Two Cayley strings differing in at most one character are translated into trees that differ in, at most, five edges (independently of the value of n). Moreover, as $n \to \infty$, the expected number of edges that will change in a tree if we change a single character of its corresponding Cayley string is 1 (see Paulden and Smith, 2006).

In addition to being compact, unbiased and having high locality, the Dandelion code has a number of features that are of high interest.

- The encoding and decoding algorithms are efficient ($\mathcal{O}(n)$ time) and easy to code.
- If a vertex label l appears k times in a given Cayley string s, then the degree of vertex l in the tree represented by s is $k+1$. This makes trivial enforcing degree constrains or forcing a fixed set of leaves on a given Cayley string, without explicitly decoding it.

All these desirable features make the Dandelion code the best candidate to represent spanning trees in population based heuristics. Furthermore, the bounded locality of the Dandelion code allows us going one step forward and define a new family of neighborhoods, which constitute a practical alternative to the k-edge-exchange family of neighborhoods traditionally used in Network Design and Routing problems.

For this we propose an extension of the Dandelion code, in which neighborhoods are represented by strings, which also have length $n-2$ but which are allowed to contain one further symbol that we denote by "_", in addition to the usual node labels $1, 2, \ldots, n$. The additional symbol has to be understood as a *wildcard* which can be assigned any possible value in $\{1, \ldots, n\}$. For example, when $n = 7$, the extended string $(3, 4, \_, 6, 7)$ should be interpreted as the neighborhood of the solution space defined by the set of elements: $\{(3, 4, \mathbf{1}, 6, 7)$, $(3, 4, \mathbf{2}, 6, 7)$, $(3, 4, \mathbf{3}, 6, 7)$, $(3, 4, \mathbf{4}, 6, 7)$, $(3, 4, \mathbf{5}, 6, 7)$, $(3, 4, \mathbf{6}, 6, 7)$, $(3, 4, \mathbf{7}, 6, 7)\}$. Similarly, for $n = 5$, the extended string $(\_, 3, \_)$ corresponds to the set of solutions: $\{(\mathbf{1}, 3, \mathbf{1})$, $(\mathbf{1}, 3, \mathbf{2})$, $(\mathbf{1}, 3, \mathbf{3})$, $\ldots$, $(\mathbf{5}, 3, \mathbf{3})$, $(\mathbf{5}, 3, \mathbf{4})$, $(\mathbf{5}, 3, \mathbf{5})\}$. Note that the length of an extended string determines uniquely the available labels. In particular, an extended string of length l can use the symbols $\_, 1, 2, \ldots, l-1, l, l+1$, and $l+2$.

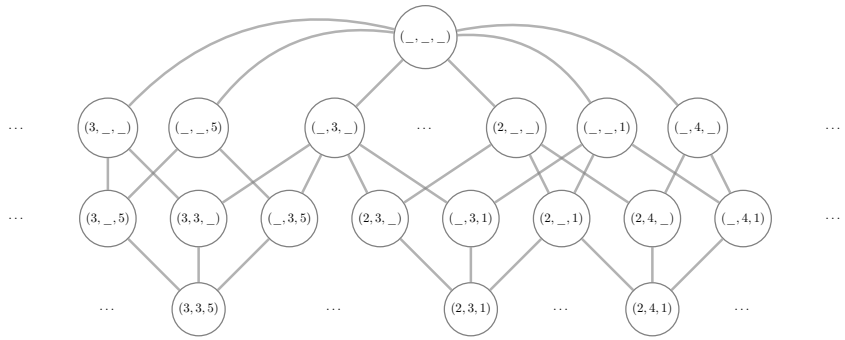


Figure 4.6: PONS of length $l = 3$

The set of neighborhoods defined by the extended strings of a given (fixed) length with the *inclusion* relation, spans a PONS, which can be represented by a directed acyclic graph in which the only node at the top represents the whole solution space, and the nodes at lower levels represent neighborhoods contained in the neighborhoods represented by the nodes just above of them (see Figure 4.6). In particular, the n^{n-2} nodes at the lowest level represent neighborhoods containing a single solution.

As n grows, the probability of obtaining a perfect mutation (changing a single character in the string changes a single edge of the tree) tends to 1 and the Dandelion neighborhoods become increasingly similar to the k-edge-exchange

neighborhoods but, nonetheless, they retain a number of features that make them interesting:

- A neighborhood with $k > 0$ wildcards contains n^k trees.
- The union of all neighborhoods differing in just k wildcard symbols from a given Cayley string contains $\mathcal{O}(n^{2k})$ trees and is called the k-character-exchange neighborhood of the string.
- It is trivially easy to obtain an uniform sample of a Dandelion neighborhoods by simply replacing all wildcards with vertex labels selected uniformly at random. It is equally easy to obtain an uniform sample from the k-character-exchange neighborhood of a Cayley string.
- The PONS allows us to alternate between exploration and exploitation by simply moving to a bigger or smaller neighborhoods along the inclusion relationships.

In relation to the last two features we propose a new improvement heuristic for the OCSTP called the Monte Carlo PONS Search (MCPS) inspired by the Monte Carlo Tree Search, which is a successful technique in the field of Board Game Artificial Intelligence (a comprehensive bibliography about it can be found in Browne et al. (2011)).

The Monte Carlo PONS Search

The Monte Carlo Pons Search (MCPS) can be seen as a Variable Neighborhood Search guided by a series of Monte Carlo trials. The algorithm performs a fixed number of iterations and each iterations can be summarized as follows:

1. Select the PONS node representing the best solution T^* obtained so far. Then, recursively select the most promising parent node of the PONS until an unexplored node N is reached.
2. For each of the parent nodes of N, evaluate an uniformly random sample of the solutions contained in the neighborhood.
3. Remember the best solution obtained in each parent node of N and update both, the best value found so far and the number of visits, for all nodes in

the path from T^* to N. Update T^* if a tree with better communication cost has been found in this iteration.

It is not necessary (neither practical) to store the whole PONS in memory from scratch. However each visited node should be stored in memory together with the two pieces of information that allow us to select the most promising nodes each iteration: an estimated value based on simulation results (Val) and the number of times it has been visited (Vis). In particular, we recursively select the parent node that minimizes the score $Val + \lambda\sqrt{\frac{Vis}{Vis_0+1}}$ where λ is a normalization factor and Vis_0 is the number of times the current node has been visited.

Although the algorithm may seem to rely heavily on the size of the sample used to evaluate each node of the PONS, most of the time it works like a local search algorithm exploring the 1-character-exchange neighborhood of T^*'s Cayley string. Only when it finds a local optimum of such neighborhood, the MCPS starts looking at bigger neighborhoods, where a sensible selection of the sample size may play a relevant role.

The main drawback of this algorithm in comparison to the AMLS is that there is no simple way to evaluate all the solutions of the 1-character-exchange neighborhood of a given Cayley string without evaluating individually each of them. Therefore, exploring the $\mathcal{O}(n^2)$ elements of a 1-character-exchange neighborhood may require $\mathcal{O}(n^4)$ time whereas Ahuja-Murty's technique is able to explore a roughly equivalent neighborhoods in just $\mathcal{O}(n^3)$ time. In exchange, the MCPS provides a more flexible framework, being able to scape from local optima using the information gathered in previous iterations to guide the search. Unfortunately, our computational experiments reveal that the trade-off between both factors favors the efficiency and further developments are required in order to obtain a competitive PONS exploration strategy.

4.3 Computational experience

In order to test empirically the heuristics described in this chapter, we have run a series of computational experiments with two sets of instances. Contreras set appeared Contreras et al. (2010a) and comprises 20 instances, whereas Random Euclidean is a set of 120 instances that we created (see Appendix B for a detailed description).

We implemented all the heuristics using the C programming language with no third-party libraries. All experiments ran in a single thread on a Windows 7 environment. The hardware used was a Dell mobile workstation with an Intel Core i7 2.50GHz processor and 16GB of RAM.

Table 4.1 summarizes the results of our first computational experiment, whose objective was to test the relative performance of several constructive heuristics. Each row contains the average optimality gap and the number of optimal (or best-known) solutions obtained with instances of the same size (Contreras set contains 4 instances of each size whereas the RE set contains 10 instances of each size). The optimality gaps of each instance were computed as $\frac{|HEUR-OPT|}{OPT}$, where OPT is the optimal value if available and the value of the best solution known otherwise. Al results were obtained in less than 0.5 seconds per instance.

Columns **Stars**, **MST** and **MCT** contain, respectively, the average gaps of the best star-shaped tree, the MST obtained with the Prim's algorithm and the MCT obtained with Gusfield algorithm. Star-shaped trees and MCTs are optimal solutions to the particular cases of the OCSTP described in Section 3.1.2, whereas MSTs have been reported as good starting solutions for evolutionary algorithms (Rothlauf, 2009b). We observe that star-shaped trees provide the most robust results with respect to the instance size, which suggest that, for larger instances, the internal structure of a spanning tree might be more important than the length of its edges.

Finally, columns **AMT** and **D&C** contain, respectively, the average gaps of the standard two-phase heuristic described in Ahuja and Murty (1987) and our Divide & Conquer heuristic described in Section 4.1. Specifically, we implemented the Bottom-up version of the Divide & Conquer heuristic that, after each merge operation, applies the AMLS to the resulting subtree. Here, we observe that both algorithms are comparable and systematically find near-optimal solutions, and

Set	|V|	Stars		MST		MCT		AMT		D&C	
		gap	best	gap	best	gap	best	gap	best	gap	best
Con	10	12.18%	0/4	17.5%	1/4	20.33%	0/4	0.22%	3/4	0.19%	3/4
Con	20	13.14%	0/4	6.57%	0/4	33.86%	0/4	0.32%	3/4	0.85%	1/4
Con	30	13.45%	0/4	18.26%	0/4	35.06%	0/4	0.0%	4/4	0.28%	2/4
Con	40	12.25%	0/4	21.22%	0/4	75.45%	0/4	0.64%	2/4	0.25%	1/4
Con	50	10.29%	0/4	22.13%	0/4	36.85%	0/4	0.03%	3/4	0.27%	3/4
RE	15	11.14%	0/10	8.88%	2/10	53.84%	0/10	0.27%	9/10	0.27%	9/10
RE	20	13.54%	0/10	12.68%	0/10	74.68%	0/10	0.17%	9/10	1.19%	5/10
RE	25	12.29%	0/10	13.31%	0/10	70.61%	0/10	0.51%	8/10	0.46%	5/10
RE	30	13.69%	0/10	14.13%	0/10	72.58%	0/10	0.01%	9/10	1.19%	7/10
RE	35	12.65%	0/10	17.2%	0/10	80.23%	0/10	0.27%	7/10	0.51%	6/10
RE	40	11.56%	0/10	22.19%	0/10	48.95%	0/10	0.23%	5/10	0.59%	4/10
RE	50	11.12%	0/10	21.21%	0/10	54.05%	0/10	0.03%	8/10	0.16%	4/10
RE	60	9.08%	0/10	26.96%	0/10	49.06%	0/10	0.56%	6/10	0.95%	2/10
RE	70	9.5%	0/10	26.49%	0/10	39.28%	0/10	0.48%	5/10	0.32%	4/10
RE	80	8.76%	0/10	30.97%	0/10	41.78%	0/10	0.64%	5/10	0.27%	5/10
RE	90	8.5%	0/10	42.42%	0/10	54.03%	0/10	0.49%	6/10	0.31%	5/10
RE	100	7.48%	0/10	34.74%	0/10	47.25%	0/10	0.64%	3/10	1.51%	2/10

Table 4.1: Computational results of constructive heuristics.

we conclude that both constructive heuristics can be used to obtain high quality solutions with small computational burden.

In order to test the improvement heuristics described in Section 4.2 we conducted another computational experiment whose results are summarized in Table 4.2.

In this second experiment we compare the computational results obtained by the AMLS and the MCPS described in Section 4.2. Specifically, we implemented the MCPS that performs 1000 Monte Carlo trials each iteration and stops after 200 iterations without improving the current best solution (with a time limit of 3600 seconds). In the experiment, both local searches were feeded with the best star-shaped tree because we wanted to measure their capacity of improving medium quality solutions and star-shaped trees provided them consistently. We also feeded these local searches with the best solution found by the constructive heuristics.

Table 4.2 contains the averaged optimality gaps, the number of optimal (or best known) solutions found and the averaged CPU times obtained with these improvement heuristics. Regarding the capacity of improving star-shaped solutions, we observe that the MCPS is able to obtain better solutions than the AMLS for medium sized instances at expenses of a higher computational cost. This capacity, however, degrades quickly as the number of vertices increases. We interpret this fact as a signal that the parameters that define the behavior of the MCPS (the stopping criterion and the number of Monte Carlo trials per iteration) must be selected according to the instance size and complexity. When both local searches are feeded with the best solution found by the constructive heuristics the MCPS performs slightly better than the AMLS. This is provably due to the fact that the best solution found by the constructive heuristics usually is a local optimum with respect to the 1-edge-exchange neighborhood and the MCPS is able to escape such local optima whereas the AMLS is not.

Set	|V|	Star+AMLS			Star+MCPS			Best Cons+AMLS			Best Cons+MCPS		
		gap	best	time	gap	best	time	gap	best	time	gap	best	time
Con	10	0.47%	3/4	0.00	0.00%	4/4	0.01	0.00%	4/4	0.00	0.00%	4/4	0.01
Con	20	0.77%	1/4	0.00	0.23%	3/4	0.91	0.00%	4/4	0.00	0.00%	4/4	0.59
Con	30	0.02%	3/4	0.00	0.09%	3/4	7.35	0.00%	4/4	0.00	0.00%	4/4	5.91
Con	40	0.56%	1/4	0.01	0.52%	1/4	63.06	0.04%	3/4	0.00	0.00%	4/4	38.10
Con	50	0.09%	3/4	0.02	0.09%	3/4	166.99	0.00%	4/4	0.00	0.00%	4/4	132.94
RE	15	0.00%	10/10	0.00	0.39%	6/10	0.14	0.00%	10/10	0.00	0.00%	10/10	0.11
RE	20	0.85%	6/10	0.00	0.17%	8/10	1.03	0.14%	9/10	0.00	0.14%	9/10	0.53
RE	25	0.19%	6/10	0.00	0.18%	7/10	2.79	0.00%	10/10	0.00	0.00%	10/10	2.16
RE	30	0.03%	8/10	0.00	0.16%	8/10	11.16	0.00%	10/10	0.00	0.00%	10/10	6.11
RE	35	0.07%	5/10	0.01	0.08%	7/10	19.46	0.03%	8/10	0.00	0.03%	8/10	15.39
RE	40	0.04%	6/10	0.01	0.29%	5/10	40.50	0.00%	9/10	0.00	0.03%	9/10	32.86
RE	50	0.21%	3/10	0.01	0.03%	8/10	253.59	0.00%	10/10	0.00	0.00%	10/10	129.41
RE	60	0.24%	1/10	0.03	0.25%	4/10	747.88	0.00%	9/10	0.00	0.02%	9/10	400.50
RE	70	0.10%	3/10	0.07	0.09%	6/10	1312.22	0.01%	9/10	0.00	0.00%	10/10	997.26
RE	80	0.14%	4/10	0.16	0.12%	5/10	3385.49	0.01%	9/10	0.00	0.01%	8/10	3312.85
RE	90	0.35%	3/10	0.25	0.51%	2/10	3600.00	0.00%	9/10	0.00	0.00%	10/10	3600.00
RE	100	0.13%	4/10	0.39	0.28%	3/10	3600.00	0.03%	7/10	0.00	0.03%	8/10	3600.00

Table 4.2: Computational results of improvement heuristics.

Chapter 5

Benders decomposition for the OCSTP

In this chapter we propose an exact algorithm for general instances of the OCSTP. Our approach is based on the Benders reformulation of the path-based formulation described in Section 3.3.2. We begin this chapter reviewing such reformulation (Section 5.1) and then propose an efficient Branch & Cut algorithm that exploits several properties of this new OCSTP formulation (Section 5.2).

5.1 Benders reformulation of the path-based formulation

Since our previous attempts to find a compact linear formulation able to deal with medium size instances of the OCSTP turned out unsuccessful, we decided to focus again on the path-based formulation, which naturally provides good linear relaxation bounds. We need, however, to find an alternative solution technique for this formulation able to deal with bigger instances and Benders Decomposition turned out to be well suited to exploit the particular structure of this formulation.

As usual in network design problems, Benders Decomposition starts by fixing the design variables to some particular feasible value $\bar{x}_{ij}$. By doing so, we obtain the following primal subproblem.

Primal Subproblem: $[\mathbf{PS}(\bar{\mathbf{x}})]$

$$\text{Min} \quad \sum_{r \in R} w^r \sum_{(i,j) \in A} c_{ij} y_{ij}^r \tag{5.1a}$$

$$\text{s.t.} \sum_{i:(i,d^r) \in A} y_{id^r}^r = 1 \qquad \forall\, r \in R \tag{5.1b}$$

$$\sum_{i:(i,j) \in A} y_{ij}^r - \sum_{k:(j,k) \in A} y_{jk}^r = 0 \qquad \forall\, r \in R \quad \forall\, j \in V \setminus \{o^r, d^r\} \tag{5.1c}$$

$$- \sum_{k:(o^r,k) \in A} y_{o^r k}^r = -1 \qquad \forall\, r \in R \tag{5.1d}$$

$$y_{ij}^r + y_{ji}^r \leq \bar{x}_{ij} \qquad \forall\, r \in R \quad \forall\, \{i,j\} \in E \tag{5.1e}$$

$$y_{ij}^r \geq 0 \qquad \forall\, r \in R \quad \forall\, (i,j) \in A \tag{5.1f}$$

The previous subproblem can be decomposed into a series of $|R|$ independent subproblems indexed by the communication requirements $r \in R$. We can do so because the objective function is the sum of the communication costs of each problem and there is not any constraint that links variables associated with different communication requirements.

Primal Subproblem r: $[\mathbf{PS_r}(\bar{\mathbf{x}})]$

$$\text{Min} \quad \sum_{(i,j) \in A} c_{ij} y_{ij}^r \tag{5.2a}$$

$$\text{s.t.} \sum_{i:(i,d^r) \in A} y_{id^r}^r = 1 \tag{5.2b}$$

$$\sum_{i:(i,j) \in A} y_{ij}^r - \sum_{k:(j,k) \in A} y_{jk}^r = 0 \qquad \forall\, j \in V \setminus \{o^r, d^r\} \tag{5.2c}$$

$$- \sum_{k:(o^r,k) \in A} y_{o^r k}^r = -1 \tag{5.2d}$$

$$y_{ij}^r + y_{ji}^r \leq \bar{x}_{ij} \qquad \forall\, \{i,j\} \in E \tag{5.2e}$$

$$y_{ij}^r \geq 0 \qquad \forall\, (i,j) \in A \tag{5.2f}$$

Note that we have eliminated the w^r factor from the objective function (it will reappear in the objective function of the Master Problem). The reasons will be clear once we look at the dual form of the subproblems:

Dual Subproblem r: $[\mathbf{DS_r}(\bar{\mathbf{x}})]$

$$\text{Max} \quad \beta^r_{d^r} - \beta^r_{o^r} - \sum_{(i,j)\in E} \bar{x}_{ij}\gamma^r_{ij} \tag{5.3a}$$

$$\text{s.t.} \quad \beta^r_j - \beta^r_i - \gamma^r_{ij} \leq c_{ij} \qquad \forall\,(i,j)\in E \tag{5.3b}$$

$$\beta^r_i - \beta^r_j - \gamma^r_{ij} \leq c_{ij} \qquad \forall\,(i,j)\in E \tag{5.3c}$$

$$\gamma^r_{ij} \geq 0 \qquad \forall\,(i,j)\in E \tag{5.3d}$$

Now it is evident that if we handle the communication requirements in the Master Problem, all subproblems share a common feasible region:

$$\mathbb{DP} = \left\{ \begin{array}{l|ll} \beta \in \mathbb{R}^{|V|} & |\beta_i - \beta_j| - \gamma_{ij} \leq c_{ij} & \forall\,(i,j)\in E \\ \gamma \in \mathbb{R}^{|E|} & \gamma_{ij} \geq 0 & \forall\,(i,j)\in E \end{array} \right\} \tag{5.4}$$

Indeed, $\mathbb{DP}$ does not depend on r and its extreme points set, $\mathcal{V}$, are shared by all the subproblems. Note that, as long as we impose the SEC in the Master Problem, all $[\mathbf{PS_r}(\bar{\mathbf{x}})]$ will be feasible (even for fractional values of the $\bar{x}_{ij}$ variables) and therefore the $[\mathbf{DS_r}(\bar{\mathbf{x}})]$ will be bounded and no extreme rays will be found ($\mathcal{R} = \varnothing$). These latter observations left us with the following Master Problem:

Master Problem: $[\mathbf{M}_\infty]$

$$\text{Min} \quad \sum_{r\in R}\sum_{(i,j)\in A} w^r d^r \tag{5.5a}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} x_{ij} = |V| - 1 \tag{5.5b}$$

$$\sum_{\{i,j\}\in E(S)} x_{ij} \leq |S| - 1 \qquad \forall\, S \subset V \tag{5.5c}$$

$$d^r \geq \bar{\beta}^r_{d^r} - \bar{\beta}^r_{o^r} - \sum_{(i,j)\in E} \bar{\gamma}^r_{ij} x_{ij} \qquad \forall\, r \in R \quad \forall\,(\bar{\beta}^r_j, \bar{\gamma}^r_{ij}) \in \mathcal{V} \tag{5.5d}$$

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

$$d^r \geq 0 \qquad \forall\, r \in R \quad \forall\,(i,j) \in A \tag{5.5f}$$

5.2 Algorithmic refinements

The Benders Reformulation $[\mathbf{M}_\infty]$ described above has two exponential families of constraints (SEC and BOC) that cannot be included in the formulation in polynomial time. Fortunately, it is not necessary to include them all to converge to the optimal solution. The classical Benders Decomposition algorithm iterates over a series of relaxed Master Problems (starting with $\mathbf{M_0}$) that does not include any SEC or BOC) to get a tentative master solution $(\bar{x}_{ij}, \bar{d}^r)$ that is used to separate up to $|V|$ SECs or up to $|R|$ new BOCs. These new constraints are then added to the Master Problem and the process continues until no violated SEC or BOC can be separated from a master solution $(\bar{x}^*, \bar{d}^*)$, which will be the optimal solution of the problem.

In order to use the $[\mathbf{M}_\infty]$ formulation, we must clearly define the separation procedure used to separate both, the SECs and the BOCs. The first family of constraints is well known and several separation procedures have been proposed in the literature. For integer values of the design variables, we can compute the connected components $(S_1, S_2, \ldots, S_k)$ induced by $\{\{i,j\} \in E \mid \bar{x}_{ij} = 1\}$. If there are more than a single connected component, at least one of such subsets generates a violated SEC, otherwise all SEC are satisfied.

SECs can also be separated for fractional values of the design variables by computing the MCT with respect to the edge capacities $\bar{x}_{ij}$. If any of the weights associated with the edges of this MCT is smaller than 1, we can derive a violated SEC from at least one of the subsets of vertices that results from removing that edge from the MCT. If the equivalent flow that traverses all the edges of the MCT is at least 1 all SECs are satisfied.

5.2.1 Separation of optimality cuts

The BOCs of $[\mathbf{M}_\infty]$ can be separated by solving the dual subproblems $[\mathbf{DS_r}(\bar{\mathbf{x}})]$ for both, integer or fractional values of $\bar{x}$. It is fundamental to note that being able to solve these $|R|$ subproblems independently speeds up their solution since it is often faster to solve $\mathcal{O}(n^2)$ linear programs of $\mathcal{O}(n^2)$ size than a single $\mathcal{O}(n^4)$ sized problem. Furthermore, since their primal counter parts $[\mathbf{PS_r}(\bar{\mathbf{x}})]$ are particular cases of the Min-Cost-Flow Problem, several highly specialized algorithms can be used to solve them efficiently (including the Network Simplex algorithm that

we use in out computational experiments). To obtain a BOC, however, we need to compute the value of the dual variables $(\beta_i^r, \gamma_{ij}^r)$ and not all Min-Cost-Flow algorithms provide them explicitly.

In fact, any integer $\bar{x}_{ij}$ that satisfies all SECs must be a tree T, so $[\mathbf{PS_r}(\bar{\mathbf{x}})]$ can be solved by finding the unique path, over T, that connects o^r and d^r. Thus, a simple Breath First Search over T lets us recover the primal subproblem solution y_{ij}^r from which we can derive a corresponding dual solution of the form:

$$\beta_i^r = \alpha_{o^r,i}^T \qquad \forall\, i \in V \quad \forall\, r \in R \tag{5.6a}$$

$$\gamma_{ij}^r = \max\{0,\ \alpha_{ij}^T - c_{ij}\} \qquad \forall\, (i,j) \in E \quad \forall\, r \in R \tag{5.6b}$$

Where the α_{ij}^T are the distance between i and j over T and all of them can be computed in just $\mathcal{O}(n^2)$ total time using the adjacency list representation of T. Now if we substitute the previous equations into the $[\mathbf{M}_\infty]$ and aggregate the BOCs we obtain an aggregated reformulation of the problem:

Aggregated Master Problem:

$$\text{Min} \quad z \tag{5.7a}$$

$$\text{s.t.} \quad \sum_{\{i,j\}\in E} x_{ij} = |V| - 1 \tag{5.7b}$$

$$\sum_{\{i,j\}\in E(S)} x_{ij} \leq |S| - 1 \quad \forall\, S \subset V \tag{5.7c}$$

$$z \geq \sum_{r\in R} w^r \alpha_{o^r d^r}^T - \sum_{(i,j)\in E} \left(\sum_{r\in R} w^r \max\{0, \alpha_{ij}^T - c_{ij}\} \right) x_{ij} \forall\, T \in \mathcal{T}_n \tag{5.7d}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, \{i,j\} \in E \tag{5.7e}$$

$$z \geq 0 \tag{5.7f}$$

Since each SEC and each aggregated BOC are defined by $\mathcal{O}(n^2)$ coefficients (and the RHS term) and all these values can be computed in just $\mathcal{O}(n^2)$ time from a integer master solution, this aggregated reformulation of the OCSTP is optimal with respect to the separation of these constraints. Moreover, the aggregated BOCs are expressed in terms that are closely related to the geometric nature

of the problem: the value z of the objective function is lower bounded by the communication cost of a given tree $\sum_{r \in R} w^r \alpha^T_{o^r d^r}$ minus the potential savings of adding another edge $\sum_{(i,j) \in E} \left(\sum_{r \in R} w^r \max\{0, \alpha^T_{ij} - c_{ij}\} \right) x_{ij}$. These savings are systematically overestimated for any particular tree, since the number of edges is fixed and adding one edge implies removing another one (thus increasing some other communication cost). However, when considered as a whole, this set of linear lower bounds correctly model the objective function, since each aggregated BOC evaluates correctly its associated tree.

Regardless of the interesting theoretical properties mentioned above, the aggregated reformulation of the OCSTP it is ill-suited for practical applications because it requires many iterations to converge to an optimal solution and the complexity of the aggregated Master Problem quickly escalates, requiring each iteration larger amounts of time as more BOCs are added. To solve such issues we must realize that there are many ways to derive an optimal dual solution from the (unique) primal solution of each subproblem. Each of these different dual solutions of the subproblems produces BOCs of different quality, which affects the overall efficiency of the algorithm.

Although the dual solutions proposed above are *natural* and can be computed efficiently, the resulting BOCs are weak and, thus, we must find a better procedure to generate dual solutions. If we are willing to sacrifice the natural interpretation of the γ^r_{ij} coefficients as *savings*, we may slightly improve the previous BOC with this alternative definition:

$$\gamma^r_{ij} \;=\; \max\{0,\; |\beta^r_i - \beta^r_j| - c_{ij}\} \qquad \forall\, (i,j) \in E \quad \forall\, r \in R \tag{5.8}$$

which provides the smallest possible value for these coefficients for a given value of the β's. Note that, since the γ^r_{ij} appear with negative coefficients in the master problem, minimizing its value provides stronger BOCs.

Unfortunately, if we solve the $[\mathbf{PS_r}(\bar{\mathbf{x}})]$ using an efficient Min-Cost-Flow algorithm (such as the Network Simplex), then recover the value of β^r_i as the dual value associated with the network balance constraints (5.2b)–(5.2d) and finally compute the value of γ^r_{ij} as defined in (5.8), the resulting BOCs are still too weak to solve efficiently instances of just 20 vertices.

5.2.2 Separation of Pareto-optimal optimality cuts

Magnanti and Wong (1981) proposed a generic separation methodology aimed at generating strong BOC. Their variant of the standard Benders algorithm requires the use of *core points* (points in the relative interior of the linear relaxation polyhedron of the Master Problem). Core points are difficult to find in general but in the OCSTP case there is a distinguished core point that is easy to compute:

$$\mathring{x}_{ij} \;=\; \frac{2}{n} \qquad \forall\, \{i,j\} \in E \tag{5.9}$$

Indeed, assuming that our underlying network is the complete graph of n vertices, $G = K_n$, we can deduce by an argument of symmetry that each edge is equally represented in the set $\mathcal{T}_n$ of all spanning trees of G. Thus, if we average all the feasible solutions of the problems the resulting *central core point* will have the same value in all the edges and since there are $\binom{n}{2} = \frac{n(n+1)}{2}$ edges and their coefficients must sum $n-1$, each of them must be equal to $\frac{2}{n}$.

The method proposed by Magnanti and Wong allows us to select, among all possible dual solutions corresponding to a given primal solution, the ones that form a Pareto-optimal family of cuts (i.e. no cut of the family is dominated by any other cut of the family). To achieve this, we must solve an alternative Dual Subproblem:

Magnanti-Wong Dual Subproblem: [**MWDS**($\bar{\mathbf{x}}$)]

$$\text{Max} \quad \sum_{r\in R} \beta^r_{d^r} - \beta^r_{o^r} - \sum_{(i,j)\in E} \mathring{x}_{ij}\gamma^r_{ij} \tag{5.10a}$$

$$\text{s.t.} \quad \sum_{r\in R} \beta^r_{d^r} - \beta^r_{o^r} - \sum_{(i,j)\in E} \bar{x}_{ij}\gamma^r_{ij} \;=\; \sum_{r\in R} w^r \alpha^T_{o^r d^r} \tag{5.10b}$$

$$\beta^r_j - \beta^r_i - \gamma^r_{ij} \;\le\; c_{ij} \qquad \forall\, r \in R \quad \forall\, (i,j) \in E \tag{5.10c}$$

$$\beta^r_i - \beta^r_j - \gamma^r_{ij} \;\le\; c_{ij} \qquad \forall\, r \in R \quad \forall\, (i,j) \in E \tag{5.10d}$$

$$\gamma^r_{ij} \;\ge\; 0 \qquad \forall\, r \in R \quad \forall\, (i,j) \in E \tag{5.10e}$$

Note that the constraint (5.10b), which ensures that the optimal solution of [**MWDS**($\bar{\mathbf{x}}$)] is also an optimal solution of [**DS**($\bar{\mathbf{x}}$)], links all commodities and does not allow us to decompose the [**MWDS**($\bar{\mathbf{x}}$)] into a set of $|R|$ independent

subproblems (although one may be tempted to naturally split this equation into $|R|$ equations of the form $\beta^r_{d^r} - \beta^r_{o^r} - \sum_{(i,j)\in E} \bar{x}_{ij}\gamma^r_{ij} = w^r \alpha^T_{o^r d^r}$ and then solve each subproblem independently and, in fact, this heuristic works well in practice).

In addition, it is necessary to know in advance the optimal value of the problem $\sum_{r\in R} w^r \alpha^T_{o^r d^r}$, which is easy to compute for integer values of the design variables but require the solution of another subproblem for fractional values of $\bar{x}_{ij}$. Furthermore, the Min-Cost-Flow structure is now lost and less efficient algorithms should be used to obtain optimal solutions of this problem.

All three issues can be solved if we use the variation of the Magnanti-Wong methodology proposed in Papadakos (2008). In its paper, Papadakos proposed to drop the additional equation but to keep the alternative objective function.

Papadakos Dual Subproblem r: $[\mathbf{PDS_r}(\bar{\mathbf{x}})]$

$$\text{Max} \quad \beta^r_{d^r} - \beta^r_{o^r} - \sum_{(i,j)\in E} \mathring{x}_{ij}\gamma^r_{ij} \tag{5.11a}$$

$$\text{s.t.} \quad \beta^r_j - \beta^r_i - \gamma^r_{ij} \le c_{ij} \qquad \forall\,(i,j)\in E \tag{5.11b}$$

$$\beta^r_i - \beta^r_j - \gamma^r_{ij} \le c_{ij} \qquad \forall\,(i,j)\in E \tag{5.11c}$$

$$\gamma^r_{ij} \ge 0 \qquad \forall\,(i,j)\in E \tag{5.11d}$$

Now, the problem decomposes again into a set of $|R|$ subproblems, each of them can be solved using a specialized Min-Cost-Flow solver (rather than a general purpose Linear Programming solver) and it is not necessary to compute the optimal value of the regular subproblem in advance. However, the Papadakos subproblem no longer depends on the master solution $\bar{x}$, and a new core point must be used each iteration to avoid generating the same BOC each iteration. To solve this issue (and taking into account that core points are difficult to find in general) Papadakos proposed a simple methodology to build a new core point from a previous core point and a feasible solution:

$$\mathring{x}^{u+1}_{ij} = \frac{\mathring{x}^{u}_{ij} + \bar{x}^{u+1}_{ij}}{2} \qquad \forall\,\{i,j\}\in E \tag{5.12}$$

Thus, averaging the previous core point with the current Master Problem solution provides the required new core point.

Several additional observations must be done about Papadakos methodology.

The SEC no longer guarantee the feasibility of the subproblem and, even if the subproblem is feasible, we cannot guarantee that a violated BOC is generated each iteration. However, the algorithm does converge and the explanation is simple: if no violated cut is produced in a given iteration, the next master solution will be equal to the previous one and the new core point will move closer and closer to that master solution. In the limit, it will reach that solution and a standard BOC will be produced, ensuring thus the convergence of the algorithm. In practice, however, it is advisable to directly generate a regular BOC each time the Papadakos subproblem fails to deliver a violated cut.

Regarding the infeasibility of the Papadakos subproblems some authors proposed alternative updating schemes for the core point that ensures feasible solutions each iteration (Martins de Sá et al., 2013). However, this procedure still does not ensure the production of a violated cut and we have found more practical to reset the core point to its initial central core point value each time an infeasible subproblem is found. Note that a core point $\mathring{x}_{ij}$ leads to an infeasible Papadakos subproblem if and only if the network with capacities $\mathring{x}_{ij}$ has a minimum cut of value less than 1. The smallest cut of the network corresponding to the central core point has a value of $\frac{2(n-1)}{n}$ and, thus, satisfies the feasibility requirements comfortably. However, after few iterations, this value may drop under the threshold of the infeasibility for some communication requirements (specially if many similar master solutions are used to update the core point) and a new core point reset must be performed.

5.2.3 Fractional cuts and filtering strategies

Once the technical issues associated with Papadakos methodology are solved, the separation of both, SECs and BOCs, can be done efficiently. However, since each subproblem generates up to $\mathcal{O}(n^2)$ new constraints, the Master Problem may grow faster than it is desirable and become unbearable after few iterations. To reduce the computational burden required to solve the Master Problem we only add cuts that are violated by the current master solution. Furthermore, instead of using the Iterative Benders algorithm, where many integer Master Problems must be solved to optimality before the algorithm converges, it is advisable to use the Branch & Cut Benders algorithm, where a single master problem is solved and both, the SECs and the BOCs, are separated dynamically.

Since we can separate SECs and BOCs in polynomial time even for fractional values of the design variables, we can perform a Fractional Warm-Start Phase before adding the integrality constraints:

1. **Base formulation:** To the relaxed Master Problem {(5.5a), (5.5b), (5.5f)}.
   (a) Add the relaxed design variable bounds: $0 \leq x_i j \leq 1$
   (b) Add the enhanced distance lower bounds: $d^i j \geq c_{ij} x_{ij} + c_{ij}^2 (1 - x_{ij})$
   (c) Add SEC for all subsets of V containing a single vertex.
   (d) Add BOC from high quality heuristic solutions.
2. **Fractional Warm-Start Phase:** Use the Iterative Benders algorithm to solve the linear relaxation of the master problem.
3. **Clean-up:** Erase all non-binding BOC generated in the previous steps.
4. **Integer Phase:**
   (a) Add the integrality constraints (5.5e).
   (b) Use the Branch & Cut Benders algorithm to solve the integer problem.

The fractional cuts generated in the Fractional Warm-Start Phase help us to ensure that the optimality gap is small enough (usually below 10%) when the Branch & Cut Benders algorithm starts branching. However, once we start solving the integer master problem, we can still separate SECs and BOCs at some fractional nodes of the Branch & Bound tree, adding only those that are largely violated by the master solution of the current node to avoid including too many constraints in the Master Poblem. We separate fractional BOCs at all fractional nodes whose depth is multiple of 5. It is tempting to filter slightly violated cuts also in integer nodes, but this will result in the algorithm incorrectly evaluating many integer solutions and returning a suboptimal tree. Notwithstanding, we can still use this technique as a Integer Warm-Start Phase:

1. **Base formulation:** To the relaxed Master Problem {(5.5a), (5.5b), (5.5f)}.
   (a) Add the relaxed design variable bounds: $0 \leq x_i j \leq 1$
   (b) Add the enhanced distance lower bounds: $d^i j \geq c_{ij} x_{ij} + c_{ij}^2 (1 - x_{ij})$

(c) Add SEC for all subsets of V containing a single vertex.

(d) Add BOC from high quality heuristic solutions.

2. **Fractional Warm-Start Phase:** Use the Iterative Benders algorithm to solve the linear relaxation of the master problem.

3. **Clean-up:** Erase all non-binding BOC generated in the previous steps.

4. **Integer Warm-Start Phase:**

   (a) Add the integrality constraints (5.5e).

   (b) Use the Branch & Cut Benders algorithm to solve the integer problem, adding only largely violated cuts.

5. **Integer Phase:** Use the Branch & Cut Benders algorithm to solve the integer problem, adding all violated cuts.

The objective of the Integer Warm-Start Phase is to provide an approximate description of the Benders reformulation polytope without clogging the Master problem with hundreds of almost-redundant constraints. For some instances, the aggressive (heuristic) filtering strategy of the Integer Warm-Start Phase might provide the optimal solution of the problem which makes the posterior Integer Phase converge immediately. For other instances, the solution provided by the Integer Phase might be incorrectly evaluated (and possibly suboptimal) due to the exclusion of a necessary BOC but, even if this is the case, all SEC and BOC generated during the Integer Warm-Start Phase are valid and constitute a tight description of the solution space, which will help the posterior Integer Phase to converge faster.

5.2.4 Local Cuts

If we are able to define local cuts that only apply to the Branch & Cut subtree rooted at a given node, it is advisable to check whether or not we can fix the value of some d^{ij} and x_{ij} variables using the following procedure:

- Let $(\bar{x}_{ij}, \bar{d}^{ij})$ the value of the master solution at the current node and let $T_1, \ldots, T_k$ the subtrees induced by $\{\{i,j\} \in E \mid \bar{x}_{ij} = 1\}$. Note that not all vertices of G must be included in $T_1 \cup \cdots \cup T_k$.

1. For all i, j such that $i \in T_l$, $j \in T_l$ and $\{i, j\} \notin T_l$ we can fix the value of the x_{ij} variable to 0 with a local cut that is valid for all descending nodes of the Branch & Cut tree.
2. For all i, j such that $i \in T_l$, $j \in T_l$ we can compute the distance between i and j over the edges of T_l and fix the value of the d^{ij} variable with a local cut that is valid for all descending nodes of the Branch & Cut tree.

Both families of local cuts might help to obtain integer solutions faster since the fixation of a single x_{ij} variable to its upper bound value (1) might trigger the generation of many of such local cuts, reducing the effective size of the problem not only in the current node but also in all the descending nodes of the Branch & Cut tree. These local cuts can be separated by inspection in just $\mathcal{O}(\sum |T_l|^2) = \mathcal{O}(n^2)$ time once the connected components $T_1, \ldots, T_k$ and their associated distance matrices, have been computed (which also requires $\mathcal{O}(n^2)$ operations).

5.2.5 Rounding Heuristic

Another useful technique that might boost the Integer Phase consists of implementing a rounding callback that provides integer feasible solutions closely related to the fractional optimal solution of a given Branch & Cut node. For the OCSTP, we can simply build the Maximum Spanning Tree with respect to the edge costs $\bar{x}_{ij}$, which will be feasible and will include all of the edges whose design variables have been already fixed to 1 and none of the edges whose design variables have been already fixed to 0. Indeed, the rounding callback can be improved further using a simple variant of the AMLS that takes into account such fixed variables to avoid removing an edge whose design variable has been fixed to 1 or adding and edge whose design variable has been already fixed to 0. This rounding procedure ensures that feasible integer solutions are found quickly and speeds up the convergence of the Integer Phase.

5.3 Computational experience

In order to test empirically our Branch & Bound algorithm, we have run a series of computational experiments.

We implemented our algorithms using the C programming language and the CPLEX 12.62 callable library. All experiments ran in a single thread on a Windows 7 environment. The hardware used was a Dell mobile workstation with an Intel Core i7 2.50GHz processor and 16GB of RAM.

We used several sets of Benchmark instances from the OCSTP literature (see Appendix B for a detailed description). Tables 5.1 and 5.2 summarize the computational results obtained with our Branch & Cut algorithm. We compare the optimality gap (computed as $\frac{UB-LB}{LB}$) and the CPU times (in seconds) obtained by our algorithm (column Benders) with the optimality gap and CPU times obtained by CPLEX using the path-based formulation (column CPLEX) and the corresponding values reported in Contreras et al. (2010a) (column Lagrangian) for a sophisticated Lagrangian Relaxation algorithm able to produce both, tight bounds and high quality solutions. This latter algorithm produces, to the best of our knowledge, the tightest upper and lower bounds for general OCSTP instances reported to the date. Moreover, the solutions obtained with this algorithm often reach (or even improve) the previously best-known value for these benchmark instances.

The first two columns of both tables determines the type and the size of each instance (we differentiate between same sized instances with a lower case letter). The third column contains the communication request density. Instances with high communication request density tend to be harder to solve in general as reported in Contreras et al. (2010a). The next two columns contain, respectively, the best lower bound and the best upper bound known so far. Upper bound values in boldface signal instances for which our algorithm produced a new best upper bound (i.e. the current best-known-solution). Finally, the optimality gap obtained with the path-based formulation, the Lagrangian Relaxation algorithm and our Branch & Cut algorithm are compared, as well as the CPU times required to obtain such values. Note that CPLEX with the path-based formulation was unable to provide a finite optimality gap for instances of 40 vertices and above within the given limits of memory (16GB) and time (2 hours).

The tables below show that our algorithm converged to the optimal solution for all benchmark instances with less than 40 vertices. For all other benchmark instances, our algorithm found the previous best-known-solution and tightened the optimality gap obtained with the path-based formulation and the Lagrangian Relaxation algorithm. Moreover, for five of the instances whose optimal solution is still unknown (*Raidl100* and *Contreras 40b, 50a, 50b & 50d*), our algorithm found solutions that improve the previous best-known-solution, including the instance *Raidl100* which has been erroneously reported as solved in previous publications. These new upper bounds have been highlighted in boldface in Tables 5.1 and 5.2.

Although the different nature of these algorithms makes difficult to compare computation times, we can se that, for the instances where all algorithms are able to find the optimal solution, the CPU times of our algorithm are consistently better than those required by the other algorithms. For those instance where our algorithm does not converge to the optimal solution we reported the results obtained when the limit of 7200 seconds (2 hours) is reached.

In order to expand our computational experience, we created a new set of Random Euclidean instances (see Appendix B for a detailed description). The results obtained with these instances are summarized in Table 5.3. There, we compare the effectiveness of our Branch & Cut algorithm with a standard CPLEX implementation of the path-based formulation.

We observe that the Branch & Cut algorithm is able to find optimal solution for 38 instances under the time limit of 2 hours. Moreover, the Branch & Cut algorithm provided tight optimality bounds for all instances where the optimal solution remains unknown. In contrast, CPLEX, with the path-based formulation solved 33 instances but was unable to provide a finite optimality gap for instances of 40 vertices within the given limits of memory (16GB) and time (2 hours). Again, the computational times of our algorithm are consistently better than those of CPLEX with the path-based formulation.

Set	|V|	$\frac{\|R\|}{\|E\|}$	Best Known		CPLEX		Lagrangian		Benders	
			LB	UB	gap	time	gap	time	gap	time
Berry	6	100.00%	534.00	534	0.00%	0.02	1.14%	0.30	0.00%	0.01
Berry	35	9.81%	16915.00	16915	0.00%	0.58	0.00%	0.94	0.00%	0.03
Berry	35u	9.81%	16167.00	16167	0.00%		13.39%	10.23	0.00%	0.03
Palmer	6	100.00%	6693180.00	6693180	0.00%	0.01	0.00%	0.30	0.00%	0.00
Palmer	12	100.00%	3428509.00	3428509	0.00%	3.36	3.82%	0.94	0.00%	0.40
Palmer	24	8.33%	1086656.00	1086656	0.00%	0.10	0.00%	10.23	0.00%	0.01
Raidl	10	97.78%	53674.00	53674	0.00%	0.12	0.06%	0.38	0.00%	0.02
Raidl	20	97.37%	157570.00	157570	0.00%	49.18	1.67%	9.54	0.00%	1.55
Raidl	50	100.00%	806864.00	806864	—	—	8.44%	338.17	6.63%	7200.00
Raidl	75	100.00%	1481533.04	1723715	—	—	14.05%	1687.57	13.33%	7200.00
Raidl	100	100.00%	2240597.69	**2551489**	—	—	14.51%	5204.76	12.18%	7200.00

Table 5.1: Computational results with benchmark instances.

Set	|V|	$\frac{\|R\|}{\|E\|}$	Best Known		CPLEX		Lagrangian		Benders	
			LB	UB	gap	time	gap	time	gap	time
Contreras	10a	53.33%	71156.00	71156	0.00%	0.07	0.33%	0.25	0.00%	0.00
Contreras	10b	57.78%	38059.00	38059	0.00%	0.02	0.00%	0.20	0.00%	0.00
Contreras	10c	51.11%	29113.00	29113	0.00%	0.01	0.00%	0.06	0.00%	0.00
Contreras	10d	48.89%	39197.00	39197	0.00%	0.25	0.82%	0.59	0.00%	0.06
Contreras	20a	36.84%	89474.00	89474	0.00%	120.29	4.75%	3.48	0.00%	16.60
Contreras	20b	35.79%	96333.00	96333	0.00%	47.72	1.63%	3.92	0.00%	9.94
Contreras	20c	54.74%	102505.00	102505	0.00%	53.88	3.94%	5.75	0.00%	5.71
Contreras	20d	53.68%	87452.00	87452	0.00%	3.17	0.49%	2.94	0.00%	0.30
Contreras	30a	57.47%	228247.00	228247	0.78%	7200.00	2.68%	27.07	0.00%	1184.28
Contreras	30b	57.93%	249607.00	249607	1.41%	7200.00	2.54%	28.70	0.00%	1705.12
Contreras	30c	53.79%	209062.00	209062	1.52%	7200.00	3.72%	31.85	0.00%	267.31
Contreras	30d	53.79%	219170.00	219170	0.00%	6009.36	3.23%	31.48	0.00%	253.64
Contreras	40a	53.85%	350542.00	350542	—	—	1.56%	93.20	0.00%	262.47
Contreras	40b	52.31%	286509.45	**291463**	—	—	4.82%	66.12	1.70%	7200.00
Contreras	40c	42.82%	282863.79	287198	—	—	5.30%	56.43	1.51%	7200.00
Contreras	40d	53.59%	321483.46	347715	—	—	10.21%	46.86	7.54%	7200.00
Contreras	50a	51.84%	441779.39	**458367**	—	—	5.00%	208.07	3.62%	7200.00
Contreras	50b	51.59%	472377.49	**506230**	—	—	8.66%	201.44	6.69%	7200.00
Contreras	50c	51.67%	369715.98	396966	—	—	9.40%	170.09	6.86%	7200.00
Contreras	50d	50.94%	466702.04	**497708**	—	—	8.44%	180.83	6.23%	7200.00

Table 5.2: Computational results with benchmark instances (cont).

Set	|V|	|R|/|E|	Best Known		CPLEX		Benders	
			LB	UB	gap	time	gap	time
RE	15a	100%	1972939.00	1972939.00	0.00%	0.65	0.00%	0.08
RE	15b	100%	989881.00	989881.00	0.00%	0.86	0.00%	0.08
RE	15c	100%	1277105.00	1277105.00	0.00%	0.24	0.00%	0.05
RE	15d	100%	1359398.00	1359398.00	0.00%	0.33	0.00%	0.06
RE	15e	100%	1351673.00	1351673.00	0.00%	0.46	0.00%	0.07
RE	15f	100%	1467683.00	1467683.00	0.00%	3.81	0.00%	0.31
RE	15g	100%	1945563.00	1945563.00	0.00%	7.32	0.00%	0.76
RE	15h	100%	1378088.00	1378088.00	0.00%	10.62	0.00%	1.50
RE	15i	100%	1281752.00	1281752.00	0.00%	1.90	0.00%	0.20
RE	15j	100%	1304410.00	1304410.00	0.00%	2.85	0.00%	0.25
RE	20a	59%	1330571.00	1330571.00	0.00%	0.97	0.00%	0.10
RE	20b	52%	1013432.00	1013432.00	0.00%	41.77	0.00%	5.64
RE	20c	59%	1466450.00	1466450.00	0.00%	10.32	0.00%	0.67
RE	20d	57%	1164643.00	1164643.00	0.00%	0.56	0.00%	0.09
RE	20e	53%	1125497.00	1125497.00	0.00%	111.90	0.00%	19.64
RE	20f	49%	1315767.00	1315767.00	0.00%	58.17	0.00%	25.92
RE	20g	56%	747806.00	747806.00	0.00%	30.52	0.00%	2.99
RE	20h	59%	1825902.00	1825902.00	0.00%	137.81	0.00%	52.39
RE	20i	58%	1227597.00	1227597.00	0.00%	5002.02	0.00%	2404.59
RE	20j	57%	1122726.00	1122726.00	0.00%	147.86	0.00%	30.70
RE	25a	54%	1174871.00	1174871.00	1.90%	7200.00	0.00%	1688.99
RE	25b	49%	1098474.00	1098474.00	0.00%	215.44	0.00%	34.82
RE	25c	59%	1402633.77	1438161.00	5.88%	7200.00	2.47%	7200.00
RE	25d	62%	1474686.00	1474686.00	0.00%	24.16	0.00%	1.92
RE	25e	55%	2048460.00	2048460.00	0.00%	596.59	0.00%	129.25
RE	25f	56%	1589238.00	1589238.00	0.00%	199.70	0.00%	14.11
RE	25g	55%	1404961.00	1404961.00	0.00%	28.08	0.00%	0.95
RE	25h	56%	1213844.00	1213844.00	0.00%	351.22	0.00%	36.81
RE	25i	61%	2363499.00	2363499.00	0.00%	300.75	0.00%	13.56
RE	25j	58%	1441073.00	1441073.00	0.00%	158.23	0.00%	11.49
RE	30a	51%	1940439.00	1940439.00	0.00%	588.22	0.00%	34.47
RE	30b	57%	2097143.95	2176098.00	7.87%	7200.00	3.63%	7200.00
RE	30c	53%	1810865.00	1810865.00	0.00%	1701.90	0.00%	93.86
RE	30d	58%	2289389.00	2289389.00	0.00%	59.34	0.00%	2.08
RE	30e	60%	1639782.69	1644400.00	2.14%	7200.00	0.28%	7200.00
RE	30f	57%	2378356.00	2378356.00	2.57%	7200.00	0.00%	1409.94
RE	30g	51%	2103005.07	2159059.00	7.69%	7200.00	2.60%	7200.00
RE	30h	60%	1883947.00	1883947.00	0.00%	375.77	0.00%	3.87
RE	30i	57%	2152307.00	2152307.00	0.00%	6273.49	0.00%	522.48
RE	30j	55%	2125764.91	2193545.00	7.61%	7200.00	3.09%	7200.00
RE	35a	55%	3621617.03	3655932.00	3.06%	7200.00	0.94%	7200.00
RE	35b	55%	2451645.00	2451645.00	1.32%	7200.00	0.00%	1357.56
RE	35c	57%	2378461.54	2425059.00	4.83%	7200.00	1.92%	7200.00
RE	35d	57%	2477704.00	2477704.00	2.86%	7200.00	0.00%	6428.12
RE	35e	57%	2743061.22	2785410.00	4.20%	7200.00	1.52%	7200.00
RE	35f	59%	2418924.00	2418924.00	2.11%	7200.00	0.00%	5467.11
RE	35g	55%	3331612.38	3635655.00	27.41%	7200.00	8.36%	7200.00
RE	35h	55%	1879525.63	1934364.00	6.23%	7200.00	2.83%	7200.00
RE	35i	60%	2353854.80	2384533.00	8.43%	7200.00	1.29%	7200.00
RE	35j	56%	2569231.04	2600285.00	6.38%	7200.00	1.19%	7200.00
RE	40a	57%	3148329.27	3468868.00	—	—	9.24%	7200.00s
RE	40b	59%	3353637.37	3634574.00	—	—	7.73%	7200.00s
RE	40c	54%	3447319.08	3643423.00	—	—	5.38%	7200.00s
RE	40d	54%	3587306.31	3887359.00	—	—	7.72%	7200.00s
RE	40e	53%	2962071.66	2973328.00	—	—	0.38%	7200.00s
RE	40f	55%	3023801.41	3218471.00	—	—	6.05%	7200.00s
RE	40g	55%	2742135.75	3055173.00	—	—	10.25%	7200.00s
RE	40h	53%	3074328.50	3243307.00	—	—	5.21%	7200.00s
RE	40i	57%	2845897.39	2906273.00	—	—	2.08%	7200.00s
RE	40j	54%	2512591.91	2593930.00	—	—	3.14%	7200.00s

Table 5.3: Computational results with new benchmark instances.

Chapter 6
Conclusions

The OCSTP is a challenging combinatorial optimization problem with application in the design of telecommunication and transportation networks. In spite of its apparent simplicity, it is very difficult to find optimal solutions to OCSTP instances, even if these instances are of moderate size.

In this dissertation, we studied several linear integer programming formulations for the OCSTP and a Branch & Cut algorithm based on the Benders reformulation of one of such formulations has been proposed. This Branch & Cut algorithm is able to obtain, in reasonable computation times, solutions that match or improve the best solutions known for several sets of instances from the OCSTP literature.

We also proposed two new combinatorial lower bounds, developed several families of valid inequalities for a previously known formulation and considered a new compact linear integer programming formulation for the OCSTP.

Regarding heuristic algorithms, we extended the AMLS to a restricted subset of the 2-edge-exchange neighborhood and proposed a new family of spanning tree neighborhoods based on the Dandelion code, which is a compact representation of spanning trees with a remarkable locality bound. We also developed a new Divide & Conquer constructive heuristic for the OCSTP and a improvement heuristic that exploits the structural properties of the Dandelion Neighborhoods. Both algorithms are comparable with the widely used Ahuja-Murty tree-building and tree-improvement heuristics.

All these results have given rise to the publication Fernández et al. (2013c) and the communications Contreras et al. (2015a,b); Fernández and Luna-Mota (2012a,b, 2014); Fernández et al. (2013a,b).

Finally, there are several topics related with the OCSTP that may be further investigated but have been left out of the scope of this dissertation. A very promising line of research consists of exploring how much the branching rule proposed in the enumerative procedure of Ahuja and Murty (1987) will improve our Benders reformulation of the OCSTP. Preliminary experiments show that it is worth to use a more sophisticated branching rule for instances of 25 vertices and above.

In a more speculative way, we consider interesting the study of the dual nature of spanning trees (that can be interpreted as a set of $n-1$ acyclic edges or as a set of $n-1$ non-crossing cuts) in relation with the OCSTP, whose objective function may benefit from both, the use of shorter edges and cuts of minimum capacity. It is possible that an hybrid linear integer programming formulation that uses *flow* and *distance* variables, can exploit the redundancies derived from the combined use of these variables to obtain better results than the flow-based and the rooted-tree formulations that we have studied.

Appendix A

The Dandelion Code

In this Appendix we introduce the details of the Dandelion Code. In particular, we present linear time algorithms for encoding and decoding Cayley strings with the Dandelion Code. Both algorithms have been taken from Paulden and Smith (2006) and are reproduced with minor variations. We encourage the reader to read the original reference in order to fully grasp the nature of the Dandelion Code and its properties.

A.1 A $\mathcal{O}(n)$ encoding algorithm

The input of Algorithm A.1 is an spanning tree with n vertices labeled as $1, 2, \ldots, n$, and its output is the corresponding *Dandelion Code* $(c_2, c_3, \ldots, c_{n-1})$.

Algorithm A.1

1. Assume that the spanning tree is rooted in the vertex labeled as n and perform a breadth-first search in order to obtain the *successor* of every vertex.

2. Using the information obtained in the previous step, find the unique path π in the tree between the vertices 1 and n as $(1, succ(1), succ(succ(1)), \ldots, n)$.

3. Break π in *cycles* as follows:

   (a) Traverse the path π from right to left and place a vertical separation mark to the immediate right of each right-to-left minimum.

   (b) Now, π can be split in several disjoint cycles $\{Z_1, \ldots, Z_t\}$ using the separation marks.

4. Recover the Dandelion Code $c_i = \phi_D(i) \quad \forall i \in (2 \ldots n-1)$, which is characterized by this 2 properties:

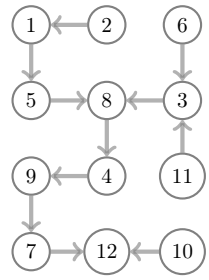
   (a) If a label l belongs to one of the cycles created in the previous step, Z_i, then $\phi_D(l)$ is the label that follows l in Z_i.

   (b) If a label l does not belong to any cycle then $\phi_D(l) = succ(l)$.

Provided that we use the adjacency list representation for the spanning tree, all the steps of the algorithm can be performed in $\mathcal{O}(n)$ time and, hence, the whole algorithm is also $\mathcal{O}(n)$.

Example: We use the spanning tree represented below to illustrate the encoding process:

1	→	2	5	
2	→	1		
3	→	6	8	11
4	→	8	9	
5	→	1	8	
6	→	3		
7	→	9	12	
8	→	3	4	5
9	→	4	7	
10	→	12		
11	→	3		
12	→	7	10	



1. Given the adjacency list representation of the spanning tree, we perform a breadth-first search starting at vertex 12 and obtain the *successor* of each other vertex:

vertex	1	2	3	4	5	6	7	8	9	10	11	12
succesor	5	1	8	9	8	3	12	4	7	12	3	–

2. Now we can easily find the unique path π in the tree between the vertices 1 and 12:

$$(1, succ(1), succ(succ(1)), \ldots, 12) = (1, 5, 8, 4, 9, 7, 12)$$

3. Break π in *cycles*:

   (a) Traverse the path π from right to left and place a vertical separation mark to the immediate right of each right-to-left minimum.

   $$(1 \mid 5, 8, 4 \mid 9, 7 \mid 12)$$

   (b) Split π in several disjoint cycles $\{Z_1, \ldots, Z_t\}$ using the separation marks.

   $$(1)\ (5, 8, 4)\ (9, 7)\ (12)$$

4. Finally, recover the Dandelion Code $c_i = \phi_D(i) \quad \forall i \in (2 \ldots n-1)$, which is characterized by these two properties:

   (a) If a label l belongs to one of the cycles created in the previous step Z_i then $\phi_D(l)$ is the label that follows l in Z_i.

   $$(\_, \_, 5, 8, \_, 9, 4, 7, \_, \_)$$

   (b) If a label l does not belong to any cycle then $\phi_D(l) = succ(l)$.

   $$(1, 8, 5, 8, 3, 9, 4, 7, 12, 3)$$

A.2 A $\mathcal{O}(n)$ decoding algorithm

The input of Algorithm A.2 is a *Dandelion Code* string $(c_2, c_3, \ldots, c_{n-1})$ and its output is the corresponding spanning tree with n vertices labeled as $1, 2, \ldots, n$.

Algorithm A.2

1. Define a function $\phi_D : [1, n] \to [1, n]$ such that:
   (a) $\phi_D(1) = 1$
   (b) $\phi_D(n) = n$
   (c) $\phi_D(i) = c_i \quad \forall i \in (2 \ldots n-1)$.
2. Let $\{Z_1, \ldots, Z_t\}$ the cycles of the function ϕ_D.
3. Rearrange the cycles in the only possible way such that:
   (a) The smallest element of every cycle, s_i, is in the rightmost place of the cycle.
   (b) The smallest elements of every cycle are in increasing order, i.e. $s_i < s_j$ when $i < j$.
4. Form a single list π with the rearranged cycles from the first element of Z_1 through the last element of Z_t.
5. Recover the spanning tree T with these final steps:
   (a) Start with n isolated vertices labeled $1, 2, \ldots, n$.
   (b) Build a path from 1 to n adding the edges: (π_1, π_2), (π_2, π_3) $\ldots$, (π_{k-2}, π_{k-1}), (π_{k-1}, π_k).
   (c) For every label l not present in the path of the previous step, add the edge: (l, c_l).

Again, since all the steps of the algorithm can be performed in $\mathcal{O}(n)$ time, the whole algorithm is also $\mathcal{O}(n)$.

Example: Following with the previous example we are going to decode $(1, 8, 5, 8, 3, 9, 4, 7, 12, 3)$ into the spanning tree of Figure A.1:

1. Define a function $\phi_D : [1, n] \to [1, n]$ such that:

   (a) $\phi_D(1) = 1$

   (b) $\phi_D(n) = n$

i	1	2	3	4	5	6	7	8	9	10	11	12
$\phi_D(i)$	1	_	_	_	_	_	_	_	_	_	_	12

   (c) $\phi_D(i) = c_i \quad \forall i \in (2 \dots n-1)$.

i	1	2	3	4	5	6	7	8	9	10	11	12
$\phi_D(i)$	1	1	8	5	8	3	9	4	7	12	3	12

2. Let $\{Z_1, \dots, Z_t\}$ the cycles of the function ϕ_D.

$$\{(12), (9, 7), (8, 4, 5), (1)\}$$

3. Rearrange the cycles in the only way such that:

   (a) The smallest element of every cycle s_i is in the rightmost place of the cycle.

$$\{(12), (9, 7), (5, 8, 4), (1)\}$$

   (b) The smallest elements of every cycle are in increasing order, i.e. $s_i < s_j$ when $i < j$.

$$\{(1), (5, 8, 4), (9, 7), (12)\}$$

4. Form a single list π with the rearranged cycles from the first element of Z_1 through the last element of Z_t.

$$(1, 5, 8, 4, 9, 7, 12)$$

5. Recover the spanning tree T applying the following final steps:

   (a) Start with n isolated vertices labeled $1, 2, \ldots, n$.

vertex	1	2	3	4	5	6	7	8	9	10	11	12
neighbors												

   (b) Build a path from 1 to n adding the edges: (π_1, π_2), (π_2, π_3) $\ldots$, (π_{k-2}, π_{k-1}), (π_{k-1}, π_k).

vertex	1	2	3	4	5	6	7	8	9	10	11	12
neighbors	5			8	1		9	5	4			7
				9	8		12	4	7			

   (c) For every label l not present in the path of the previous step, add the edge: (l, c_l).

vertex	1	2	3	4	5	6	7	8	9	10	11	12
neighbors	5	1	6	8	1	3	9	5	4	12	3	7
	2		8	9	8		12	4	7			10
			11					3				

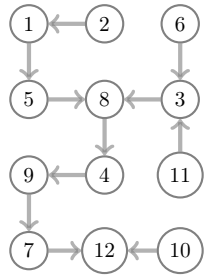


Figure A.1: Directed Spanning Tree rooted at vertex 12.

Appendix B

OCSTP Instances

In the computational experiments reported in Section 3.2.2, Section 3.3.4, Section 4.3 and Section 5.3, we used several sets of Benchmark instances from the OCSTP literature. In particular we obtained from Contreras a set of eleven well-known instances already used in Rothlauf (2006) and another set of twenty instances used in Contreras et al. (2010a) for the first time:

1. **Palmer instances:** (Palmer, 1994) Three instances of 6, 12 and 24 vertices representing cities in the United States of America. The distances are obtained from a database whereas the communication requests are inversely proportional to the distances.

2. **Raidl instances:** Five instances attributed to Raidl with 10, 20, 50, 75 and 100 vertices. Distances and communication requirements are uniformly distributed in the range $[0, 100]$.

3. **Berry instances:** (Berry et al., 1995) Three instances of 6, 35 and 35 vertices. The bigger instances have the same communication requirements but one of them (*Berry35u*) has been modified to made all distances equal.

4. **Contreras instances:** (Contreras, 2009) Twenty instances of 10, 20, 30, 40 and 50 vertices (four of each size) derived from *Raidl100* instance by randomly selecting subsets of vertices and imposing that about 50% of the communication requirements must be 0.

Additionally, we created a new set of 120 Random Euclidean instances which are available from the author upon request:

5. **Random Euclidean:** 120 instances divided in 12 groups of 10 instances with 15, 20, 25, 30, 35, 40, 50, 60, 70, 80, 90 and 100 vertices respectively. For all these instances the distances are the metric closure of a cost matrix whose entries are uniformly distributed in the range [1,100] whereas the communication requirements are uniformly distributed in the range [1,1000]. Additionally, for instances with 20 or more vertices, a 44% of the communication requests, randomly chosen, have been set to 0 to obtain instances that are comparable with the Contreras set.

Finally, we used the information obtained by our Branch & Cut algorithm to learn more about the structure of the optimal solutions. In Table B.1 we report the average path length (i.e. the average number of edges in the path connecting two vertices), the diameter (i.e. the maximum number of edges in the path connecting two vertices) and the degree distribution of the optimal solutions (or best known solutions if optimal solutions were unavailable) of several sets of Random Euclidean instances (we report the average values for 10 instances of each size). We also report the corresponding values of the MST associated to the cost matrix c of these instances as well as corresponding values of stars, binary trees and paths of the same size.

We observe that OCST and stars tend to have small average path lengths, as expected. OCST, however, differ from stars when we compare their diameter or their degree distribution. In the first aspect, OCST are similar to binary trees, whose diameter is proportional to $\log_2(|V|)$. Regarding de degree distribution, however, OCST are more similar to MST but tend to have more vertices of high degree. These observations are consistent with the findings of other authors (see Rothlauf, 2009b; Steitz and Rothlauf, 2008, 2009, 2012a) and with the nature of the OCSTP.

	|V|	Path Length	Diam.	Degree Distribution 1	2	3	4	5	6	7	8	9	10	...	|V − 1|
Stars	15	1.87	2.0	93.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	6.7%
	20	1.90	2.0	95.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	5.0%
	25	1.92	2.0	96.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	4.0%
	30	1.93	2.0	96.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	3.3%
	35	1.94	2.0	97.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	2.9%
	40	1.95	2.0	97.5%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	2.5%
OCST	15	1.63	6.7	52.0%	23.3%	14.7%	7.3%	1.3%	1.3%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	20	1.78	7.0	56.0%	18.0%	13.5%	8.5%	1.5%	1.5%	1.0%	0.0%	0.0%	0.0%	...	0.0%
	25	1.93	7.8	58.4%	18.8%	9.6%	5.2%	4.0%	2.4%	1.2%	0.0%	0.4%	0.0%	...	0.0%
	30	2.16	8.6	52.0%	23.3%	11.3%	7.3%	4.7%	1.3%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	35	2.23	9.5	56.0%	19.1%	10.9%	7.1%	3.7%	2.0%	0.9%	0.3%	0.0%	0.0%	...	0.0%
	40	2.34	10.0	56.3%	18.8%	12.5%	5.8%	2.5%	2.5%	0.8%	0.4%	0.4%	0.0%	...	0.0%
MST	15	3.55	7.5	44.0%	32.0%	18.7%	4.0%	1.3%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	20	4.34	9.4	43.0%	33.5%	15.5%	7.0%	0.5%	0.5%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	25	4.76	10.8	44.4%	32.4%	14.0%	6.0%	2.4%	0.8%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	30	5.32	12.5	44.0%	30.3%	16.3%	7.3%	1.7%	0.3%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	35	5.85	13.8	39.4%	35.7%	18.3%	4.6%	1.7%	0.3%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	40	6.50	15.0	42.8%	31.8%	16.3%	7.0%	1.8%	0.3%	0.3%	0.0%	0.0%	0.0%	...	0.0%
Binary Trees	15	3.5	6.0	53.3%	6.7%	40.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	20	4.06	8.0	50.0%	10.0%	40.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	25	4.51	8.0	52.0%	4.0%	44.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	30	4.91	8.0	50.0%	6.7%	43.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	35	5.26	10.0	51.4%	2.9%	45.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	40	5.53	10.0	50.0%	5.0%	45.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
Paths	15	5.33	14.0	13.3%	0.0%	13.3%	73.3%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	20	7.00	19.0	10.0%	0.0%	10.0%	80.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	25	8.67	24.0	8.0%	0.0%	8.0%	84.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	30	10.33	29.0	6.7%	0.0%	6.7%	86.7%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	35	12.00	34.0	5.7%	0.0%	5.7%	88.6%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%
	40	13.67	39.0	5.0%	0.0%	5.0%	90.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	...	0.0%

Table B.1: Structural analysis of the optimal or best known solutions of Random Euclidean instances.

Index

Bibliography

Ahuja, R. K. and V. V. S. Murty (1987). Exact and heuristic algorithms for the optimum communication spanning tree problem. *Transportation Science 21*(3), 163–170. [Cited on pages 24, 25, 47, 52, 62, and 86].

Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik 4*, 238–252. [Cited on page 16].

Berry, L. T. M., B. A. Murtagh, and G. McMahon (1995). Applications of a genetic-based algorithm for optimal design of tree-structured communication networks. In *Proceedings of the Regional Teletraffic Engineering Conference of the International Teletrafficc Congress*, pp. 361–370. [Cited on page 93].

Browne, C., P. Rohlfshagen, and E. Powley (2011). Monte-carlo tree search. http://www.mcts.ai/. [Cited on page 60].

Bulajich Manfrino, R., J. A. Gómez Ortega, and R. Valdez Delgado (2009). *Inequalities - A Mathematical Olympiad Approach*. Birkhäuser Basel. [Cited on page 30].

Cayley, A. (1889). A theorem on trees. *Quarterly Journal of Mathematics 23*, 376–378. [Cited on page 9].

Contreras, I. (2009). *Network Hub Location: Models, Algorithms, and Related Problems*. Ph. D. thesis, Universitat Politècnica de Catalunya. [Cited on pages 25, 32, and 93].

Contreras, I. and E. Fernández (2012). General network design: A unified view of combined location and network design problems. *European Journal of Operational Research 219*(3), 680–697. [Cited on pages 3 and 24].

Contreras, I., E. Fernández, and C. Luna-Mota (2015a, June). Benders decomposition for the OCSTP. In CORS/INFORMS Conference, Montréal, Canada. [Cited on pages 4 and 85].

Contreras, I., E. Fernández, and C. Luna-Mota (2015b, October). Benders decomposition for the OCSTP. In Workshop on Combinatorial Optimization, Routing and Location, Salamanca, Spain. [Cited on pages 4 and 85].

Contreras, I., E. Fernández, and A. Marín (2010a). Lagrangean bounds for the optimum communication spanning tree problem. *TOP 18*(1), 140–157. [Cited on pages 25, 41, 42, 44, 62, 79, and 93].

Contreras, I., E. Fernández, and A. Marín (2010b). The tree-of-hubs location problem: A comparison of formulations. *European Journal of Operational Research 202*, 390–400. [Cited on pages 3 and 24].

Cormen, T. H., C. E. Leiserson, R. L. Rivest, and C. Stein (2009). *Introduction to Algorithms* (3rd ed.). MIT Press. [Cited on pages 2 and 8].

Eisner, J. (1997). State-of-the-art algorithms for minimum spanning trees: A tutorial discussion. Manuscript available online (78 pages), University of Pennsylvania. [Cited on pages 7 and 11].

Fernández, E. and C. Luna-Mota (2012a, August). A compact formulation for the OCSTP. In International Symposium on Mathematical Programming, Belin, Germany. [Cited on pages 4 and 85].

Fernández, E. and C. Luna-Mota (2012b, May). A compact formulation for the OCSTP. In Workshop on Combinatorial Optimization, Routing and Location, Benicassim, Spain. [Cited on pages 4 and 85].

Fernández, E. and C. Luna-Mota (2014, July). A dandelion code extension: using codes beyond genetic algorithms. In 20th IFORS Conference, Barcelona, Spain. [Cited on pages 4 and 85].

Fernández, E., C. Luna-Mota, A. Hildenbrandt, G. Reinelt, and S. Wiesberg (2013a, January). A 3-index formulation for the OCSTP. In 17th Combinatorial Optimization Workshop, Aussois, France. [Cited on pages 4 and 85].

Fernández, E., C. Luna-Mota, A. Hildenbrandt, G. Reinelt, and S. Wiesberg (2013b, May). A flow formulation for the OCSTP. In International Network Optimization Conference, Tenerife, Spain. [Cited on pages 4 and 85].

Fernández, E., C. Luna-Mota, A. Hildenbrandt, G. Reinelt, and S. Wiesberg (2013c). A flow formulation for the optimum communication spanning tree. *Electronic Notes in Discrete Mathematics 41*, 85–92. [Cited on pages 4, 25, and 85].

Fischer, T. (2007). *Improved local search for large optimum communication spanning tree problems*, pp. 2–4. 7th Metaheuristics International Conference. [Cited on pages 25 and 54].

Fischer, T. and P. Merz (2007). A memetic algorithm for the optimum communication spanning tree problem. In *Hybrid Metaheuristics*, Volume 4771 of *Lecture Notes in Computer Science*, pp. 170–184. Springer. [Cited on pages 25 and 54].

Fischetti, M., G. Lancia, and P. Serafini (2002). Exact algorithms for minimum routing cost trees. *Networks 39*(3), 161–173. [Cited on pages 3, 24, and 25].

Ford, L. R. and D. R. Fulkerson (1956). Maximal flow through a network. *Canadian Journal of Mathematics 8*, 399–404. [Cited on page 6].

Gomory, R. E. and T. C. Hu (1961). Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics 9*(4), 551–570. [Cited on pages 12, 13, and 35].

Gottlieb, J., B. A. Julstrom, F. Rothlauf, and G. R. Raidl (2001). Prüfer numbers: A poor representation of spanning trees for evolutionary search. In *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pp. 343–350. [Cited on page 10].

Gusfield, D. (1990). Very simple methods for all pairs network flow analysis. *SIAM Journal on Computing 19*(1), 143–155. [Cited on page 13].

Hu, T. C. (1974). Optimum communication spanning trees. *SIAM Journal on Computing 3*(3), 188–195. [Cited on pages 2, 7, 21, 23, 24, and 34].

Johnson, D. S., J. K. Lenstra, and A. H. G. Rinnooy Kan (1978). The complexity of the network design problem. *Networks 8*(4), 279–285. [Cited on pages 2 and 24].

Korte, B. and J. Vygen (2007). *Combinatorial Optimization: Theory and Algorithms* (4th ed.). Springer. [Cited on page 33].

Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society 7*(1), 48–50. [Cited on page 12].

Lawler, E. L. (2001). *Combinatorial Optimization: Networks and Matroids*. Dover Publications. [Cited on pages 7 and 29].

Li, Y. and Y. Bouchebaba (2000). A new genetic algorithm for the optimal communication spanning tree problem. In *Artificial Evolution*, Volume 1829 of *Lecture Notes in Computer Science*, pp. 162–173. Springer. [Cited on page 25].

Magnanti, T. L. and L. A. Wolsey (1995). Optimal trees. In *Handbooks in Operations Research and Management Science*, Volume 7, pp. 503–615. Elsevier. [Cited on pages 32 and 33].

Magnanti, T. L. and R. T. Wong (1981). Accelerating benders decomposition: Algorithmic enhancement and model selection criteria. *Operations Research 29*(3), 464–484. [Cited on pages 19, 41, and 73].

Martí, R. and G. Reinelt (2011). *The Linear Ordering Problem: Exact and Heuristic Methods in Combinatorial Optimization*. Springer. [Cited on page 41].

Martins de Sá, E., R. Saraiva de Camargo, and G. de Miranda (2013). An improved benders decomposition algorithm for the tree of hubs location problem. *European Journal of Operational Research 226*(2), 185–202. [Cited on page 75].

Palmer, C. (1994). *An approach to a problem in network design using genetic algorithms*. Ph. D. thesis, Polytechnic University, Troy, NY. [Cited on page 93].

Palmer, C. and A. Kershenbaum (1994). Two algorithms for finding optimal communication spanning trees. Technical report, IBM T. J. Watson Research Center. [Cited on page 25].

Papadakos, N. (2008). Practical enhancements to the magnanti-wong method. *Operations Research Letters 36*(4), 444–449. [Cited on pages 19 and 74].

Papadimitriou, C. and M. Yannakakis (1988). Optimization, approximation, and complexity classes. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pp. 229–234. [Cited on pages 2 and 24].

Paulden, T. and D. K. Smith (2006). From the dandelion code to the rainbow code: a class of bijective spanning tree representations with linear complexity and bounded locality. *IEEE Transactions on Evolutionary Computation 10*(2), 108–123. [Cited on pages 10, 58, and 87].

Peleg, D. (1997). Approximating minimum communication cost spanning trees and related problems. pp. 1–11. International Colloquium on Structural Information and Communication Complexity. [Cited on page 25].

Picciotto, S. (1999). *How to encode a tree.* Ph. D. thesis, University of California, San Diego. [Cited on page 10].

Prüfer, H. (1918). Neuer beweis eines satzes über permutationen. *Archiv für Mathematik und Physik 27*, 142–144. [Cited on page 9].

Ravelo, S. V. and C. E. Ferreira (2015). Ptas for some metric p-source communication spanning tree problems. In *WALCOM: Algorithms and Computation*, Volume 8973 of *Lecture Notes in Computer Science*, pp. 137–148. Springer. [Cited on page 26].

Reinelt, G. (2013). Lower bounds for the optimum communication spaning tree problem. Personal communication. [Cited on page 28].

Reshef, E. and D. Peleg (1998). Deterministic polylog approximation for minimum communication spanning trees. pp. 670–681. International Colloquium on Automata, Languages and Programming. [Cited on page 25].

Rothlauf, F. (2006). *Representations for Genetic and Evolutionary Algorithms* (2nd ed.). Springer. [Cited on pages 26 and 93].

Rothlauf, F. (2007). Design and applications of metaheuristics. Technical report, Universität Mannheim. [Cited on page 25].

Rothlauf, F. (2009a). An encoding in metaheuristics for the minimum communication spanning tree problem. *INFORMS Journal on Computing 21*(4), 575–584. [Cited on page 26].

Rothlauf, F. (2009b). On optimal solutions for the optimal communication spanning tree problem. *Operation Research 57*(2), 413–425. [Cited on pages 24, 26, 62, and 94].

Sharma, P. (2006). Algorithms for the optimum communication spanning tree problem. *Annals of Operations Research 143*, 203–209. [Cited on page 26].

Soak, S. M. (2006). A new evolutionary approach for the optimal communication spanning tree problem. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences E89-A*(10), 2882–2893. [Cited on page 25].

Steitz, W. and F. Rothlauf (2008). Orientation matters: How to efficiently solve ocst problems with problem-specific eas. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pp. 563–570. ACM. [Cited on pages 24, 26, and 94].

Steitz, W. and F. Rothlauf (2009). New insights into the ocst problem: Integrating node degrees and their location in the graph. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pp. 357–364. ACM. [Cited on pages 24, 26, and 94].

Steitz, W. and F. Rothlauf (2012a). Edge orientation and the design of problem-specific crossover operators for the ocst problem. *IEEE Transactions on Evolutionary Computation 16*(1), 108–116. [Cited on pages 24, 26, and 94].

Steitz, W. and F. Rothlauf (2012b). Using penalties instead of rewards: Solving OCST problems with guided local search. *Swarm and Evolutionary Computation 3*, 46–53. [Cited on page 26].

Tilk, C. and S. Irnich (2015, June). Combined column-and-row generation for the optimal communication spanning tree problem. In 6th International Workshop on Freight Transportation and Logistics , Ajaccio, France. [Cited on page 25].

Wolf, S. and P. Merz (2010). Efficient cycle search for the minimum routing cost spanning tree problem. In *Evolutionary Computation in Combinatorial Optimization*, Volume 6022 of *Lecture Notes in Computer Science*, pp. 276–287. Springer. [Cited on pages 25 and 54].

Wolsey, L. A. (1998). *Integer programming*. Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & sons. [Cited on page 15].

Wu, B. (2002). A polynomial time approximation scheme for the two-source minimum routing cost spanning trees. *Journal of Algorithms 44*, 359–378. [Cited on page 26].

Wu, B., K.-M. Chao, and C. Tang (2000a). Approximation algorithms for some optimum communication spanning tree problems. *Discrete Applied Mathematics 102*, 245–266. [Cited on page 25].

Wu, B. Y., K. Chao, and C. Tang (2002). Light graphs with small routing cost. *Networks 39*(3), 130–138. [Cited on page 25].

Wu, B. Y., K.-M. Chao, and C. Y. Tang (2000b). A polynomial time approximation scheme for optimal product-requirement communication spanning trees. *Journal of Algorithms 102*, 245–266. [Cited on page 25].

Wu, B. Y., G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C. Tang (2000c). A polynomial time approximation scheme for minimum routing cost spanning trees. *SIAM Journal on Computing 29*, 761–778. [Cited on pages 3, 24, and 25].