Optimal Communication Spanning Tree

Jainor Nestor Cardenas Choque

Dissertation presented to the
Institute of Mathematics and statistics
of the
University of São Paulo

Program: Master in Computer Science

Advisor: Prof. Dr. Yoshiko Wakabayashi

During the development of this work the author received financial support from CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior), Brazil, Finance Code 001

São Paulo, March 2021

Optimal Communication Spanning Tree

Esta versão da tese contém as correções e alterações sugeridas pela Comissão Julgadora durante a defesa da versão original do trabalho, realizada em 08/07/2021. Uma cópia da versão original está disponível no Instituto de Matemática e Estatística da Universidade de São Paulo.

Comissão Julgadora:

- Profa. Dra. Yoshiko Wakabayashi (orientadora) - IME-USP
- Prof. Dr. Fábio Luiz Usberti (IC-UNICAMP)
- Prof. Dr. Phablo F.S. Moura (DCC-UFMG)

Agradecimentos

A realização desta dissertação de mestrado contou com importantes apoios aos quais serei eternamente grato. Sem esses apoios esta dissertação não teria se tornado uma realidade.

Em primeiro lugar, gostaria de agradecer a Deus por todos os momentos maravilhosos que tenho tido e por tudo que tenho superado na vida. Agradeço muito à minha família, meus pais Luis e Esperanza, e meus irmãos Miriam e Galvani. Muito obrigado por terem me apoiado em todas as minhas decisões, mesmo que isso tenha significado viver longe, em outro país.

De maneira especial gostaria de agradecer à minha orientadora, professora Yoshiko Wakabayashi, pela paciência, por todo o conhecimento transmitido, pelo enorme entusiasmo que me contagiou e pelo seu apoio em todos os momentos. Também gostaria de agradecer aos professores Carlinhos, Alexandre, e Ana Rosa; e aos membros da comissão julgadora, os professores Fábio Luiz Usberti e Phablo F.S. Moura, pelas sugestões que contribuíram para melhorar este trabalho. Além disso, gostaria de agradecer muito ao Renzo Gómez, que foi, na pratica, meu co-orientador, e me ajudou muito na realização deste trabalho.

Gostaria também de agradecer aos meus amigos da Universidade de São Paulo, Juan, Renzo, Hans, Christian, Ana, Nury, Jhon, Joseph, Carlos, Miguel, Noemi, Grover, ao pessoal do futebol e do futsal da faculdade, pelas conversas, reuniões, e pelos momentos de sossego. De forma especial, gostaria de mencionar meu grande amigo, Gabriel, com quem compartilhei muitos momentos no laboratório e fora da universidade.

Por fim, quero estender o meu agradecimento, a todos aqueles que, de um modo ou de outro, tornaram possível a realização desta dissertação.

Resumo

CARDENAS, J. N. **Árvore Geradora de Comunicação Ótima**. 2021. 85 f. Tese (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

Neste trabalho estudamos o problema da Árvore Geradora de Comunicação Ótima (AGCO). Uma instância deste problema consiste de uma quádrupla (G, c, R, w) composta por um grafo conexo $G = (V, E)$, uma função não-negativa c que atribui a cada elemento $e \in E$ um custo $c(e)$, um conjunto R de pares de vértices em V, e uma função não-negativa w, chamada demanda, definida sobre R. Cada par (u, v) de R é chamado um requisito, o vértice u é chamado origem e o vértice v é chamado destino do par. Para uma dada árvore geradora T de G, o custo de comunicação de um requisito $r = (u, v)$ é definido como a demanda $w(r)$ multiplicada pela distância entre u e v em T (sendo a distância a soma dos custos das arestas no caminho de u a v em T).

No problema da Árvore Geradora de Comunicação Ótima, dada uma instância (G, c, R, w), o objetivo é encontrar em G uma árvore geradora que minimiza a soma total dos custos de comunicação de todos os requisitos em R. Este problema foi introduzido por T. C. Hu em 1974 e é sabido ser NP-difícil. Alguns de seus casos especiais, não tão triviais, podem ser resolvidos em tempo polinomial.

Investigamos aqui dois tais casos especiais do problema AGCO, ambos para o caso de G ser um grafo completo. No primeiro deles, todas as arestas do grafo têm o mesmo custo. Neste caso, a solução é dada pela árvore de Gomory-Hu de uma certa rede associada à instância dada. No segundo problema, todos os requisitos têm a mesma demanda, e a solução é dada por uma árvore que é uma estrela.

Também estudamos algumas formulações lineares inteiras mistas para o problema AGCO. Para isso, estudamos formulações lineares para o problema da árvore geradora mínima, algumas das quais fazem uso de fluxos. Tais formulações são combinadas e dão origem a algumas formulações mistas para o problema AGCO. Implementamos algoritmos branch-and-cut para tais formulações, e apresentamos os resultados computacionais obtidos.

Palavras-chave: árvore geradora, árvore geradora de comunicação ótima, árvore Gomory-Hu, branch-and-cut, programação linear inteira.

Abstract

Cardenas, J. N. **Optimal Communication Spanning Tree Problem**. 2021. 85 f. Tese (Mestrado) - Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2021.

In this work we address the Optimal Communication Spanning Tree (OCST) problem. An instance of this problem consists of a tuple (G, c, R, w) composed of a connected graph $G = (V, E)$, a nonnegative cost function c defined on E, a set R of pairs of vertices in V, and a nonnegative function w, called demand, defined on R. Each pair (u, v) of R is called a requirement, the vertex u is called origin, and the vertex v is called destination of the pair. For a given spanning tree T of G, the communication cost of a requirement pair $r = (u, v)$ is defined as the demand $w(r)$ multiplied by the distance between u and v in T (the distance being the sum of the costs of the edges in the path from u to v).

In the Optimal Communication Spanning Tree (OCST) problem, we are given an instance (G, c, R, w) and we seek a spanning tree in G that minimizes the overall sum of the communication costs of all requirements in R. This problem was introduced by T. C. Hu in 1974 and is known to be NP-hard. Some of its special cases, not so trivial, can be solved in polynomial time.

We address two such special cases of the OCST problem, both restricted to complete graphs. The first one is the Optimum Requirement Spanning Tree (ORST) problem, in which all edges have the same cost (a constant). In this case, an optimal solution is given by a Gomory-Hu tree of a certain associated network. The second one is a special case of the OCST problem, in which all requirements have the same demand. This problem is called Minimum Routing Cost Spanning tree (MRCT) (and is also known as the Optimum Distance Spanning Tree problem).

We also study the main mixed integer linear programming (MILP) formulations for the OCST problem. For that, we first study formulations for the spanning tree problem, some purely combinatorial and some based on flows (leading to mixed formulations). Furthermore, we exhibit the computational results of the experiments we conducted with our implementation of a branch-and-cut approach for the different MILP formulations that we studied.

Keywords: spanning tree, optimal communication spanning tree, branch-and-cut, Gomory-Hu tree, integer linear program.

Contents

List of abbreviations

OCST	Optimal Communication Spanning Tree
CS	Cut Set
SEC	Subtour Elimination Constraints
MST	Minimum Spanning Tree
LP	Linear Programming
ILP	Integer Linear Programming
MILP	Mixed Integer Linear Programming
MRCT	Minimum Routing Cost Spanning Tree
ORST	Optimum Requirement Spanning Tree
PB	Path Based
FB	Flow Based
RTB	Rooted Tree Based

List of Figures

Chapter 1

Introduction

A spanning tree of a graph G is a connected subgraph of G that is acyclic and contains all of its vertices. Spanning trees are combinatorial objects that arise in many optimization problems related to network design. For example, if we are interested in constructing an electrical network, our aim is to minimize the total length of the links in the network, as the construction cost increases with each additional link. In this case, a minimum (cost) spanning tree (MST) is an optimal solution.

In other situations, the network needs to ensure a communication requirement between some pairs of vertices in the network. In such cases, we seek to minimize the operational cost of the network. In general, the operational cost depends on both, the communication requirement and the distance between pairs of vertices in the network. In this work, we focus on the Optimal Communication Spanning Tree (OCST) problem that models such situations.

In the OCST problem, we are given a tuple (G, c, R, w) composed of a connected graph $G = (V, E)$, a nonnegative cost function c defined on E, a set R of pairs of vertices in V, and a nonnegative function w, called demand, defined on R. Each pair $r = (r_o, r_d)$ of R is called a requirement, the vertex r_o is called origin, and the vertex r_d is called destination. Our objective is to find a spanning tree T of G that minimizes the overall sum of the communication cost between the pairs in R, where the communication cost of a requirement pair $r = (r_o, r_d)$ is defined as the cost of the path from r_o to r_d in T multiplied by its demand $w(r)$. This problem arises in different areas, as for example, in computational biology [WLB+00] and network design [MW84].

This text is organized as follows. In Chapter 2, we present basic concepts, notation, and terminology in graph theory, network flows, and mixed integer linear programming (MILP). In Chapter 3, we formally define the OCST problem. After that, we present a historical review of the most relevant results regarding OCST that exist in the literature. We continue with special cases of OCST that admit polynomial-time algorithms.

Later, in Chapter 4, we address different MILP formulations that have been proposed

for OCST. We begin this chapter by showing MILP formulations for MST that have an exponential number of inequalities. Since the formulations for OCST use these inequalities, we also show how to decide, in polynomial time, if a solution satisfies those inequalities. After that, we present some MILP formulations for OCST.

Finally, in Chapter 5, we show our computational results, comparing the performance of the formulations presented in the previous chapter. In Chapter 6, we present some final considerations on our study of the OCST problem, and mention some possible directions for future work.

Chapter 2

Preliminaries

In this chapter, we introduce the basic concepts, notation, and terminology that we use throughout this text. The notation and terminology that we use are conventional in graph theory, polyhedral combinatorics, and combinatorial optimization, but they are included to make the text self-contained. In each section, some references are given on some of the topics.

2.1 Graph theory

A **graph** is a pair $G = (V, E)$, where V and E are disjoint sets, together with an incidence function ψ that maps each element in E to an unordered pair of elements of V. The elements of V are called **vertices** and the elements of E are called **edges**. Given a graph G, we also denote by $V(G)$ and $E(G)$ its set of vertices and its set of edges, respectively. The **order** of a graph G is the cardinality of $V(G)$. If $e \in E$, and $\psi(e) = \{u, v\}$ (possibly $u = v$), we say that e **links** the vertices u and v, and that u and v are the **ends** of e. Furthermore, we say that e is **incident** to the vertices u and v, and that u and v are **adjacent**. In case $u = v$, the edge uv is called a **loop**. Two or more edges with the same pair of ends are said to be **parallel**.

We say that a graph is **simple** if it contains neither loops nor parallel edges. Throughout this text, in the problems to be addressed here, we always consider that the input graph is simple, but we may not state this explicitly, just in the beginning of the chapter. We will also simplify the notation, and to refer to simple graphs $G = (V, E)$, we do not specify the incidence function ψ, and consider that the edge set E consists of unordered pairs of vertices of V. Thus, we say simply that $e = \{u, v\}$ is an edge of G. For convenience, when there is no risk of confusion, instead of $\{u, v\}$, we may write uv.

A **path** in a graph G is a sequence of the form $P = \langle v_1, e_1, v_2, \ldots, e_k, v_{k+1} \rangle$, where $v_1, \ldots, v_{k+1}$ are distinct vertices of G and e_i is an edge that links v_i and v_{i+1} for $i = 1, 2, \ldots, k$. In this case, we say that P is a **path between** v_1 and v_{k+1}, or that P **links** v_1 and v_{k+1}, and that v_1 and v_{k+1} are the **ends** of P. A vertex is called an **internal** vertex of P if it

is not an end of P. The **length** of a path P, denoted by $|P|$, is its number of edges. We denote by P_{uv}, a path that links two vertices u and v in G. Usually, we represent a path P just by its sequence of vertices, writing $P = \langle v_1, \ldots, v_{k+1} \rangle$. If a graph $G = (V, E)$ has a cost function $c : E \to \mathbb{R}_{\geq 0}$ defined on E, the **cost** of a path in G is defined as the sum of the costs of its edges. The cost of a path of minimum cost between two vertices u and v in G is also called the **distance** (with respect to c) between u and v, and is denoted $\text{dist}_G(u, v)$, or simply $\text{dist}(u, v)$. In case the edges have no cost associated with them, we consider that every edge has unit cost.

Let $G = (V, E)$ and $G' = (V', E')$ be graphs such that $V' \subseteq V$ and $E' \subseteq E$. In this case, we say that G' is a **subgraph** of G, and that G' is **contained** in G. We also say that G is a **supergraph** of G'. Moreover, if $V' = V$ we say that G' is a **spanning subgraph** of G. Given $S \subseteq V$, the subgraph of G **induced** by S is the subgraph $G[S] = (S, A)$, where $A \subseteq E$ is the subset of edges of E whose ends belong to S. Furthermore, we denote by $G - S$ the subgraph $G[V \setminus S]$, and if $F \subseteq E$, we denote by $G - F$ the graph $(V, E \setminus F)$. If α is a vertex or an edge of G, we write $G - \alpha$ instead of $G - \{\alpha\}$. Given $A \subseteq E$, we denote by $G[A]$ the subgraph of G whose edge set is A, and whose vertex set consists of all ends of edges of A. The **neighborhood** of a vertex v in G is the set $N_G(v) = \{u \in V : u \text{ is adjacent to } v\}$. A vertex in $N_G(v)$ is said to be a **neighbor** of v. If G is a simple graph, the **degree** of a vertex v in G, denoted by $\deg_G(v)$, is equal to $|N_G(v)|$. (In both cases, the subscript G may be omitted when there is no danger of confusion.)

Let $G = (V, E)$ be a graph and let $S, T \subseteq V$. We denote by $E(S, T)$ the set of edges in G with one end in S and the other in T. When $T = S$, we abbreviate it to $E(S)$. If S is a nonempty proper set of V and $T = V \setminus S$, the set $E(S, T)$ is called the **edge cut** of G associated with S, and we denote it by $\delta(S)$. Note that $\delta(S) = \delta(V \setminus S)$. If we consider two distinct vertices $s, t \in V$, and a subset $S \subset V$ such that $s \in S$ and $t \notin S$, then we say that $\delta(S)$ is a s-t **cut** or that S **separates** s from t.

We say that a graph G is connected if there exists a path that links each pair of vertices in G. A **component** of G is a maximal connected subgraph of G. A **cycle** in G is a sequence of the form $C = \langle v_1, e_1, v_2, \ldots, v_k, e_k, v_{k+1} \rangle$, where $v_1, v_2, \ldots, v_k$ are distinct vertices of G, $v_1 = v_{k+1}$ and e_i links v_i to v_{i+1}, for $i = 1, 2, \ldots, k$. If G does not contain a cycle, we say that G is **acyclic**. An acyclic graph is also called a **forest**. A **tree** is a connected forest.

In a tree $T = (V, E)$, we say that a vertex v is an **internal** vertex of T if its degree is greater than one. Otherwise, we say that v is an **external** vertex or a **leaf**. An **extremal internal** vertex is an internal vertex that has at least one neighbor that is a leaf. A **star** is a tree with at most one extremal internal vertex.

A directed graph, or **digraph**, is a pair $D = (V, A)$, where V and A are disjoint sets, together with an incidence function ψ that maps each element in A to an ordered pair of elements of V. The elements of V are called **vertices** or **nodes** and the elements of A are called **arcs**. We also denote by $V(D)$ and $A(D)$ the set of vertices and the set of arcs,

respectively, of D. If $a \in A$, and $\psi(a) = (u, v)$, we say that u is the **tail** of a and that v is its **head**. We also say that u and v are the **ends** of a.

Let $D = (V, A)$ be a digraph. For $S \subseteq V$, the subgraph of D induced by S, denoted by $D[S]$, is the digraph (S, F), where $F \subseteq A$ is the subset of arcs with its two ends in S. Furthermore, we denote by $D - S$ the subgraph $D[V \setminus S]$, and if $F \subseteq A$, we denote by $D - F$ the digraph $(V, A \setminus F)$. If α is a vertex or an arc of D, we write $D - \alpha$ instead of $D - \{\alpha\}$. Let $v \in V(D)$. The **indegree** of a vertex v in D, denoted by $\deg^-_D(v)$, is the number of arcs whose head is v; and the **outdegree** of v in D, denoted by $\deg^+_D(v)$ is the number of arcs whose tail is v. Similarly to the case of graphs, concepts such as loops, parallel arcs, adjacency of vertices can be also be defined (we omit them, as we assume they are clear).

A **path** in D is a sequence of the form $P = \langle v_1, a_1, v_2, \ldots, a_k, v_{k+1} \rangle$, where $v_1, \ldots, v_{k+1}$ are distinct vertices of D and a_i is an arc with tail v_i and head v_{i+1}, for $i = 1, 2, \ldots, k$. We say that P is a path that **begins** in v_1 and **ends** in v_{k+1}, and has length k. Usually, we represent a path P just by its sequence of vertices, writing $P = \langle v_1, \ldots, v_{k+1} \rangle$.

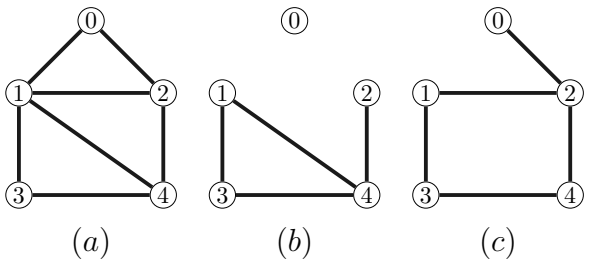


Figure 2.1: (*a*) *A graph G;* (*b*) *a subgraph of G;* (*c*) *a spanning subgraph of G.*

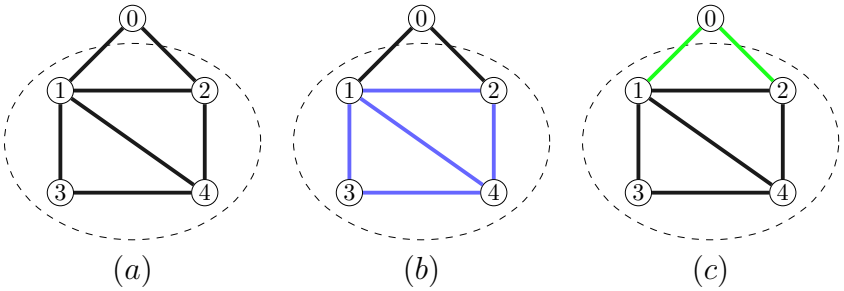


Figure 2.2: (*a*) *A graph $G = (V, E)$ and a set $S = \{1, 2, 3, 4\}$;* (*b*) *blue edges belong to $E(S)$;* (*c*) *green edges belong to $\delta(S)$.*

The **underlying graph** of a digraph $D = (V, A)$, denoted by $G(D)$, is the graph obtained from D by replacing each arc with an edge with the same ends. Conversely, any graph G can be regarded as a digraph, by replacing each edge by two opposite oriented arcs with the same ends. This digraph is called the **associated digraph** of G, and is denoted by $D(G)$.

An **orientation** of a simple graph $G = (V, E)$ is a digraph obtained from G by replacing each edge, say $\{u, v\}$, by an arc with the same ends, either (u, v) or (v, u). A **rooted tree** T is an orientation of a tree in which every vertex, except one, has indegree equal to one. The vertex with indegree zero is called the **root** of T. We also refer to a rooted tree with root r

as an r-**arborescence**. Clearly, in an r-arborescence T there exists a unique path from r to every other vertex in T. In case there exists a path from u to v, in T, we say that u is an **ancestor** of v, and that v is a **descendant** of u.

For convenience, as in the case of graphs, we refer to an arc of a digraph as a pair (u, v). When an edge (or arc) is used as a subscript, the braces (or parentheses) are always omitted. Moreover, a comma is used only when it is necessary. For instance, if edges $\{1, 2\}$ and $\{1, 12\}$ are subscripts of a vector x, we use the notation x_{12} and $x_{1,12}$, respectively.

For the readers interested in these and other concepts in graph theory, we suggest [BM08] and [Die12].

2.2 Linear and integer programming

A Linear Program (LP) is an optimization problem of the following form: given vectors $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and a matrix $A \in \mathbb{R}^{m \times n}$, find a vector $x \in \mathbb{R}^n$ that satisfies the following restrictions:

$$\text{minimize} \qquad \sum_{i=1}^{n} c_i x_i \tag{2.1}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} A_{ij} x_j \leq b_i \qquad \text{for } i = 1, \ldots, m, \tag{2.2}$$

$$x_j \geq 0 \qquad \text{for } j = 1, \ldots, n. \tag{2.3}$$

We say that x_i is a **variable** of the linear program, for $i = 1, \ldots, n$. Each linear inequality $\sum_{j=1}^{n} A_{ij} x_j \leq b_i$ is called a **constraint**. A vector $x \in \mathbb{R}^n$ that satisfies all the constraints is called a **feasible solution** of the linear program. Furthermore, we say that a feasible solution x is an **optimal solution** if for every feasible solution x' we have

$$\sum_{i=1}^{n} c_i x_i \leq \sum_{i=1}^{n} c_i x'_i.$$

We refer to Linear Programming problems as the class of problems of the above form. We note that similar maximization problems with constraints that are equalities are also included in this class. It is well known that they can be solved in polynomial time, either by the Interior Point method [Kar84] or the Ellipsoid method [Hač79]. In many applications, the Simplex method [Dan63] is the most used algorithm for solving linear programs, because of its good performance on average (although it is known that on some instances it may require exponential time).

An Integer Program (IP) is an optimization problem of the form:

$$\text{minimize} \quad \sum_{i=1}^{n} c_i x_i \tag{2.4}$$

$$\text{subject to} \quad \sum_{j=1}^{n} A_{ij} x_j \leq b_i \qquad \text{for} \quad i = 1, \ldots, m, \tag{2.5}$$

$$x_j \in \mathbb{Z}_{\geq 0} \qquad \text{for } j = 1, \ldots, n. \tag{2.6}$$

That is, it is an LP with the additional restriction that every variable must be a nonnegative integer. It is known that an integer program is an NP-hard problem [GJ79]. When all variables are required to be binary (0 or 1), it is called a 0/1 **linear program**.

We say that a vector is binary if all its entries are binary. Furthermore, we denote by $\mathbb{B}^n$ the set of binary vectors in $\mathbb{R}^n$.

Solving an integer program is equivalent to solving a linear program with potentially high number of constraints [Mey74]. Thus, we can convert any IP to an LP, so that every extreme point solution of the LP is an optimal solution for the IP. The same result is applicable for a Mixed-Integer Linear Program (MILP), where we have integer and real variables.

Many problems in combinatorial optimization can be modelled as integer programs. When we want to solve a specific integer program, say $\mathcal{I}$, one common approach is to drop the integrality constraint over its variables, and study what is called the **linear relaxation** of $\mathcal{I}$.

Note that linear relaxations may have an advantage over other schemes, because if an optimal solution for the linear relaxation is integral, then it is also an optimal solution for the former IP. In any case, this relaxation gives a lower bound for the value of an optimal solution of $\mathcal{I}$, if $\mathcal{I}$ is a minimization problem. Researchers study how to strengthen the relaxed LP by adding valid inequalities, preferably facets of the polyhedron defined by the convex hull of the feasible solutions of $\mathcal{I}$. This approach is used in the subarea known as polyhedral combinatorics. In the next sections we present some basic concepts useful in linear programming problems and polyhedral combinatorics.

2.3 Polyhedron and separation problem

Let $x_1, \ldots, x_m$ be vectors in $\mathbb{R}^n$. We say that a vector $x \in \mathbb{R}^n$ is a **linear combination** of $x_1, x_2, \ldots, x_m$ if there exist real numbers $\lambda_1, \ldots, \lambda_m$ such that $x = \sum_{i=1}^{m} \lambda_i x_i$. Moreover, a linear combination such that $\sum_{i=1}^{m} \lambda_i = 1$, for $i = 1, 2, \ldots, m$, is called an **affine combination**. Similarly, a linear combination such that $\sum_{i=1}^{m} \lambda_i = 1$ and $\lambda_i \geq 0$, for $i = 1, 2, \ldots, m$, is called a **convex combination**. A convex combination is called **proper** if $\lambda_i > 0$, for $1, \ldots, m$. We say that vectors $x_1, x_2, \ldots, x_m$ are **linearly independent** if the system $\sum_{i=1}^{m} \lambda_i x_i = \vec{0}$ has a unique solution: $\lambda_i = 0$, for $i = 1, \ldots, m$. In a similar way, vectors $x_1, x_2, \ldots, x_m$ are **affinely independent** if the system $\sum_{i=1}^{m} \lambda_i x_i = \vec{0}$, $\sum_{i=1}^{m} \lambda_i = 0$ has a unique solution $\lambda_i = 0$, for

$i = 1, \ldots, m$.

A set of vectors V in $\mathbb{R}^n$ is a **vector space** if, and only if, for each two vectors x, y in V the vector $x + y$ is in V and for any vector z in V, and a real number $\lambda \in \mathbb{R}$ the vector λz also belongs to V. That is, V is closed under sum and multiplication by a scalar.

Let V be a vector space in $\mathbb{R}^n$ and let S be any set of vectors in $\mathbb{R}^n$. If every vector in V can be expressed as a linear combination of vectors of S, we say that S **span** V. Moreover, if S is a subset of V such that it is closed under sum and multiplication by scalar, we say that S is a **subspace** of V. We denote the span of vectors $x_1, x_2, \ldots, x_m$ by $\text{span}\{x_1, x_2, \ldots, x_m\}$. Moreover, for any set of vector $x_1, x_2, \ldots, x_m$, in V, $\text{span}\{x_1, x_2, \ldots, x_m\}$ is a subspace of V. Furthermore, if S spans V and S is minimal, then S is called a **basis** of V. Moreover, the following conditions are equivalent:

- The set S is a minimal spanning set of V.
- The set S is an independent spanning set of V.
- The set S is a maximal linearly independent set of V.

So, any basis of V has the same number of vectors. Furthermore, the **dimension** of V is defined as the cardinality of any basis of V.

Let $A \in \mathbb{R}^{m \times n}$ be a matrix. The **rank** of A, denoted by $\text{rank}(A)$, is the dimension of the vector space spanned by its rows (or equivalently, columns). If A is a square matrix, we denote the **determinant** of A by $\det(A)$. Moreover, A is called **nonsingular** if $\det(A) \neq 0$. In this case there exists a unique inverse matrix A^{-1} such that $AA^{-1} = I$ where I is the identity matrix. Besides, if $\text{rank}(A) = n$, we say that A has **full rank**. Observe that A has full rank if, and only if, $det(A) \neq 0$.

A **polyhedron** P is a set of points that can be described as $\{x \in \mathbb{R}^n : Ax \leq b\}$ where A is an $m \times n$ matrix, and b is a vector in $\mathbb{R}^m$. A **hyperplane** is a polyhedron of the form $\{x \in \mathbb{R}^n : c'x = \delta\}$ for some $c \in \mathbb{R}^n$ and $\delta \in \mathbb{R}$. We say that a polyhedron P is a **polytope** if it is the convex hull of a finite set of points. That is, a polyhedron $P \subset \mathbb{R}^n$ is a polytope if, and only if, there exist vectors $l, u \in \mathbb{R}^n$ such that $l \leq x \leq u$, for all $x \in P$. A point x is called a **vertex** of a polytope $P \subseteq \mathbb{R}^n$ if it is not a proper convex combination of two (or more) distinct points in P. The vertices of a polytope can also be characterized in another equivalent way. A point $x' \in P$ is a vertex if there exists a hyperplane H such that

$$\{x'\} = P \cap H.$$

Moreover, x' satisfied with equality by $rank(P)$ linearly independent inequalities of the system $Ax \leq b$.

Given a polyhedron of the form $Q = \{(u, x) \in \mathbb{R}^p \times \mathbb{R}^q : Au + Bx \leq b\}$, where $A \in \mathbb{R}^{m \times p}$ and $B \in \mathbb{R}^{m \times q}$ and $b \in \mathbb{R}^m$, the **projection** of Q onto $\mathbb{R}^q$, or onto the x-space, is defined as

$$Proj_x(Q) = \{x \in \mathbb{R}^q : \exists u \in \mathbb{R}^p : (u, x) \in Q\}.$$

In what follows consider the polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$. If $c \in \mathbb{R}^n$ is a nonzero vector and $\delta = \min\{cx : Ax \leq b\}$, the hyperplane $\{x : cx = \delta\}$ is called a **supporting hyperplane** of P. A polyhedron F is called a **face** of P if $F = P$ or $F = P \cap H$ for some supporting hyperplane H. Thus, F is a face if $F = \{x \in P : A'x = b'\}$ for some subsystem $A'x \leq b'$ of $Ax \leq b$. A polyhedron (polytope) is called **integral** if all of its vertices have integer coordinates.

We say that a constraint $a'x \leq \beta$ is **tight** (or **active**) in a face F of P, if $a'x = \beta$ holds for each $x \in F$. The **dimension** of a polyhedron P, denoted by $dim(P)$, is the maximum number of affinely independent points in P minus one. Moreover, $dim(P) = n - rank(A)$. Thus, the empty face has dimension equal to -1, and a vertex of P has dimension equal to 0. If a face F has dimension $n-1$, it is called a **facet**.

A polyhedron $P \subset \mathbb{R}^n$ is a **valid formulation** for a set $X \subset \mathbb{Z}^n$ if $X = P \cap \mathbb{Z}^n$. A matrix $A \in \mathbb{R}^{m \times n}$ is **totally unimodular** (TU) if every square submatrix of A has determinant $-1, 0$ or 1.

Consider a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$ such that b is an integer vector. An important result in integer programming states that: if A is TU, then P is integral [CCZ14]. This result can be used to show that some network flow problems can be solved in polynomial time. It may also used to obtain some min-max results.

The importance of integer programming lies in its power to model many problems in combinatorial optimization that have a linear objective function. Well known problems such as minimum cost spanning tree of a graph, maximum matching in a graph, the travelling salesman problem can all be modelled as integer programs. In some cases, this modelling enhanced with further approaches lead to polynomial-time algorithms, in some other cases, good approximation algorithms can be obtained. Many exact methods can also take advantage of further knowledge on the polyhedron associated with the integer LP.

Some of these techniques will be used in Chapter 4. We will not go into further details here, but we refer the interested reader to [Sch03]

Given a polytope $P \subseteq \mathbb{R}^n$, and a vector $z \in \mathbb{R}^n$, the **separation problem** (with respect to P and z) consists in deciding whether z belongs to P or not; and if not, find an inequality $c^T x \leq c_o$ with $c \in \mathbb{R}^n$ such that

$$\sum_{i=1}^{n} c_i x_i \leq c_o \text{ for every } x \in P \text{ and } \sum_{i=1}^{n} c_i z_i > c_o.$$

When $z \notin P$, the inequality $c^T x \leq c_o$ is called a separating hyperplane.

Grötschel, Lovász, and Schrijver [GLS88] showed that if we can solve the separation problem in polynomial time, the corresponding linear problem (over P) can be solved in polynomial time. For further reading, we suggest [CCZ14].

2.4 Combinatorial optimization problem

A **Combinatorial Optimization Problem** (COP) can be modelled as a triple $(\mathcal{I}, \mathtt{S}, f)$, where $\mathcal{I}$ is the set of instances of the problem, $\mathtt{S}$ is the set of feasible solutions, and f is a function (called objective function) $f : \mathtt{S} \to \mathbb{R}$ that assigns a value to each feasible solution $S \in \mathtt{S}$. For $I \in \mathcal{I}$, let us denote by $\mathtt{S}(I)$ the set of all feasible solutions for I.

Let $\Pi = (\mathcal{I}, \mathtt{S}, f)$ be a COP, and let I be an instance of Π. Then, I can be described by a triple $(E, w, \mathcal{S})$, where E is a nonempty finite set, w is a function $w : E \to \mathbb{R}$, and $\mathcal{S} = \mathtt{S}(I)$ is composed by the subsets of E. For linear objective functions f, we have $f(S) = \sum_{e \in S} w_e$. Given an instance $I = (E, w, \mathcal{S})$, the goal is to find a feasible solution $S \in \mathcal{S}$ that minimizes (or maximizes) $f(S)$.

We denote by $\mathbb{R}^E$ the set of all real-valued vectors indexed by the elements of E. For each solution $S \in \mathcal{S}$, the **incidence vector** of S, denoted by $\chi^S \in \mathbb{R}^E$, is defined as:

$$\chi_e^S = \begin{cases} 1, & \text{if } e \in S, \\ 0, & \text{otherwise.} \end{cases}$$

We may associate a polyhedron $P \subseteq \mathbb{R}^{|E|}$ with the triple $(E, w, \mathcal{S})$ as being the convex hull of the incidence vectors of the solutions in $\mathcal{S}$. That is,

$$P := \text{conv}\{\chi^S \in \mathbb{R}^E : S \in \mathcal{S}\}.$$

Thus, a COP with a linear objective function can be defined as $\min\{wx : x \in P\}$. The set of vertices of P corresponds to the set of incidence vectors of solutions $S \in \mathcal{S}$. Since P is a polytope, we know that P can be described by a system of linear inequalities. Thus, we can model a COP as a 0/1 LP. Conversely, every 0/1 LP can be described as a COP.

2.5 Network flows

A **network** is a pair $N = (D, u)$ where $D = (V, A)$ is a digraph, and u is a nonnegative function (called capacity) defined on A. A **flow** in N is a nonnegative function $x : A \to \mathbb{R}_{\geq 0}$ that satisfies the following condition: $x_a \leq u_a$ for each arc $a \in A$.

If x is a flow and $R \subset V$, then $x^+(R)$ (resp. $x^-(R)$) denotes the sum of the flows on the arcs with tail (resp. head) in R and head (resp. tail) in $V \setminus R$. The **excess** of x in R is $x^+_-(R) = x^+(R) - x_-(R)$.

Given a network $N = (D, u)$ and two distinct vertices s and t in D, called **source** and **sink**, respectively, the **maximum flow problem** consists in finding a flow x in N that maximizes $x^+_-(s)$, subject to $x^+_-(v) = 0$ for every $v \in V \setminus \{s, t\}$. The value $x^+_-(s)$ is called the value of the flow x (from s to t). More formally, we can model the maximum flow problem as an LP as follows:

$$\begin{aligned} \text{maximize} \quad & \sum_{sk \in A} x_{sk} - \sum_{ks \in A} x_{ks} && (2.7) \\ \text{subject to} \quad & \\ & \sum_{ij \in A} x_{ij} - \sum_{jk \in A} x_{jk} = 0 \qquad \forall j \in V \setminus \{s,t\} \\ & x_{ij} \geq 0 \qquad \forall ij \in A \\ & x_{ij} \leq u_{ij} \qquad \forall ij \in A. \end{aligned}$$

Let $G = (V, E)$ be a graph, and let $u' : E \to \mathbb{R}_{\geq 0}$ be a function defined on E. The **network associated** with G is the pair $(D(G), u)$ such that, for each edge $ij \in E$, the capacity of the arcs $ij, ji \in A(D(G))$ is $u_{ij} = u_{ji} = u'_{ij}$.

Let (G, u) be a network and let $S \subset V(G)$ be such that $\delta(S)$ is an s-t cut. The **capacity** of $\delta(S)$ is defined as the sum of the capacities of each edge in $\delta(S)$, and its denoted by $u(\delta(S))$. Moreover, two cuts $\delta(X)$ and $\delta(Y)$, are said to **cross** (overlap) if X and Y cross (overlap).

Dantzig and Fulkerson [DF56] showed an important result in network flows, known as the **max-flow min-cut theorem**. This theorem states that, in a network (D, u) with source s and sink t, the maximum amount of flow going from s to t is equal to the capacity of a minimum cut separating them. This statement also holds for undirected graphs.

Given a network $N = (D, u)$, a cost function $c : A \to \mathbb{R}_{\geq 0}$, and a function $b : V \to \mathbb{R}$, the **minimum cost flow problem** consists in finding a flow x that minimizes cx such that $x^+_-(v) = b(v)$, for every vertex v in $V(D)$. This problem can be modelled by the following LP formulation:

$$\begin{aligned} \text{minimize} \quad & \sum_{ij \in A} c_{ij} x_{ij} && (2.8) \\ \text{subject to} \quad & \\ & \sum_{ij \in A} x_{ij} - \sum_{jk \in A} x_{jk} = b_j \qquad \forall j \in V && (2.9) \\ & x_{ij} \leq u_{ij} \qquad \forall ij \in A && (2.10) \\ & x_{ij} \geq 0 \qquad \forall ij \in A && (2.11) \end{aligned}$$

The case in which each arc ij has an infinite capacity ($u_{ij} = \infty, \quad \forall ij \in A$) is called the **transhipment problem**. For this problem, the LP formulation is as follows:

$$\text{minimize} \quad \sum_{ij \in A} c_{ij} x_{ij} \tag{2.12}$$

subject to

$$\sum_{ij \in A} x_{ij} - \sum_{jk \in A} x_{jk} = b_j \qquad \forall j \in V \tag{2.13}$$

$$x_{ij} \geq 0 \qquad \forall ij \in A \tag{2.14}$$

Observe that, the matrices arising from a maximum flow problem, a minimum cost flow problem and a transhipment problem, are totally unimodular. Therefore, the extreme points of the polyhedron associated to any of these problems are integral.

2.6 Exact algorithms to solve ILP problems

In this section we describe the main techniques we have used in the implementation of the algorithms presented in Chapter 4. We also mention briefly other techniques that may be used to solve ILP (or MILP) optimization problems.

The **branch-and-bound** (B&B) method is a fundamental and widely used scheme to solve hard combinatorial optimization problems. Let $I = (E, w, \mathcal{S})$ be an instance of a COP $\Pi = (\mathcal{I}, \mathtt{S}, f)$, such that $E = \{1, \ldots, n\}$ (see the notation we have defined in Section 2.4). The B&B method iteratively builds a binary search tree T of subproblems, or subsets of $\mathcal{S}$. The root of T represents $\mathcal{S}$, its left child represents $\mathcal{S}_0 = \{S \in \mathcal{S} : \chi_1^S = 0\}$, and its right child represents $\mathcal{S}_1 = \{S \in \mathcal{S} : \chi_1^S = 1\}$. After that, we recursively divide the subproblems. That is, the children of the left child of the root represent the sets $\mathcal{S}_{00} = \{S \in \mathcal{S}_0 : \chi_2^S = 0\}$, and $\mathcal{S}_{01} = \{S \in \mathcal{S}_0 : \chi_2^S = 1\}$, and so on. Throughout the algorithm, a feasible solution $S^* \in \mathcal{S}$, called the **incumbent solution**, is stored globally. At each iteration, the algorithm selects a new subproblem from the list of unexplored subproblems. If a feasible solution S' is found such that $f(S') < f(S^*)$, the incumbent solution is updated. When considering a subproblem, if the lower bound for the value of this subproblem is greater or equal to $f(S^*)$, the algorithm ignores this subproblem. Otherwise, the problem is divided into subproblems. Once all the subproblems had been explored, the best incumbent solution is returned.

The main idea in a **cutting plane method** is to solve an integer program by solving succesive linear programs. Let P be the polytope defined by the integer program, and let P' be its linear relaxation. A generic cutting plane method is as follows [CCZ14]:

- Find an optimal solution x^* for P'. If it is integral and $x^* \in P$. Stop.
- Otherwise, find a cutting plane (valid inequality) that separates x^* from P, and add the plane to the formulation. Repeat the step recursively.

The **branch-and-cut** method consists in applying branch-and-bound and the cutting

planes techniques at the same time. This method involves running a branch-and-bound algorithm and using cutting planes to tighten the linear programming relaxation. The latter is done by finding new cutting planes that are added to the LP relaxation at every subproblem. The technique of adding new constraints is also known as **row generation**.

When formulating a problem as an MILP, there may exist reasons for considering formulations with a huge number of variables. Sometimes a compact formulation may have a weak LP relaxation. Frequently, the relaxation can be tightened by a formulation that involves a huge number of variables. Another reason could be that a compact formulation of a MIP may have a symmetric structure that causes branch-and-bound to perform poorly because the problem barely changes after branching. A reformulation with a huge number of variables can eliminate this symmetry [BJN+98]. The disadvantage of solving these formulations in a solver is that we cannot consider all of the variables explicitly.

In a **column generation** approach, the assumption is that in an optimal solution most of the variables will be nonbasic. Therefore, only a subset of variables may be considered when solving the problem. Therefore, we add only the variables which have the potential to improve the objective function (based on their reduced costs).

Dantzig–Wolfe decomposition is an algorithm for solving linear programming problems with a special structure that relies on delayed column generation technique. In this scheme, the original formulation is reformulated into a **Master Problem** (MP). The **Restricted Master Problem** (RMP) is equal to the master problem with the difference that only a set of columns (variables) is considered. A subproblem associated with each column is considered to generate columns (to the RMP) whose inclusion improve the objective function. The problem of choosing such columns is also called **pricing problem**.

The **branch-and-price** method uses both branch-and-bound and column generation techniques simultaneously. This technique can be used for solving large-scale LP problems. Generally, the original problem is reformulated by using Dantzig-Wolfe decomposition. If this method includes row generation it is also called **branch-and-price-and-cut** method.

The technique known as **Benders decomposition** reformulates a mixed integer linear program to one with fewer variables and an exponential number of constraints, which can be separated eficiently by solving an LP subproblem, known as the **Dual Subproblem** (DSP). To accomplish this, the original formulation is projected into the discrete variable space, resulting in a reformulation known as the **Benders Master Problem** (MP). In consequence, the contribution of the continuous variables in the original formulation is estimated by two sets of constraints known as feasibility and optimality cuts indexed by the sets of the extreme rays and the extreme points of DSP, respectively. For further references in these topics, see [Wol98, MJSS16].

2.7 Approximation algorithms

Let $\Pi = (\mathcal{I}, \mathcal{S}, f)$ be a minimization problem, where $\mathcal{I}$ is the set of instances of Π, $\mathcal{S}$ is the set of all feasible solutions for each instance $I \in \mathcal{I}$, and f is the objective function.

For a given instance $I \in \mathcal{I}$, if $S \in \mathcal{S}$ is a solution for I, and S minimizes f, we say that S is an **optimal solution** for I. We denote by $\mathrm{OPT}(I)$ the value of an optimal solution for an instance $I \in \mathcal{I}$.

Let A be an algorithm that, for each $I \in \mathcal{I}$, returns a feasible solution $A(I)$. Let α be such that $\alpha \geq 1$, where α is a constant or a function of I. We say that A is an **α-approximation** for Π if it runs in polynomial time on the size of I, and

$$f(A(I)) \leq \alpha \, \mathrm{OPT}(I), \text{ for every } I \in \mathcal{I}.$$

In this case, we say that A has **approximation factor** (or ratio) α.

A **Polynomial-Time Approximation Scheme** (PTAS) for Π is a family of algorithms $\{A_\epsilon\}$, where $\{A_\epsilon\}$, is a $(1+\epsilon)$-approximation, for each $\epsilon > 0$. Thus a PTAS can be seen as an algorithm that receives, as input, an instance I and an $\epsilon > 0$. This algorithm runs in polynomial time on the size of I considering ϵ as a constant. If this algorithm also runs in polynomial time on ϵ, we say that $\{A_\epsilon\}$ is a **Fully Polynomial-Time Approximation Scheme** (FPTAS). For further reading on approximation algorithms, we suggest [WS11].

Chapter 3

Optimal communication spanning trees

In this chapter, we address the Optimal Communication Spanning Tree (OCST) problem. We define this problem formally in Section 3.1 and then, in Section 3.2, we present a historical review of the results on OCST that have appeared in the literature. In this review, we highlight the type of results that characterized each period of time. In Section 3.3, we mention some applications of the OCST problem, and in Section 3.4, we define an important structure – the Gomory-Hu tree – which is used to solve a special case of the OCST problem. We finish this chapter by focusing on two special cases of this problem that can be solved in polynomial time.

In all problems considered in this chapter, the input graph is always connected and simple.

3.1 The OCST problem

An instance of the OCST problem consists of a tuple of the form $\mathcal{I} = (G, c, R, w)$, where $G = (V, E)$ is a connected graph, $c : E \to \mathbb{R}_{\geq 0}$ is a cost function defined on the edges, $R \subset V \times V$, and $w : R \to \mathbb{R}_{\geq 0}$ is the demand function defined on R. We say that R is a set of communication requirements, and for a requirement $r \in R$, we consider that $r = (r_o, r_d)$, and refer to r_o (resp. r_d) as the origin (resp. destination) of r. Instead of $w(r)$, sometimes we write w_r.

Given such an instance $\mathcal{I} = (G, c, R, w)$, we say that a spanning tree T in G has **communication cost** $\mathrm{cc}(T)$, a measure defined as follows:

$$\mathrm{cc}(T) = \sum_{r \in R} w_r \, \mathrm{dist}_T(r_o, r_d), \tag{3.1}$$

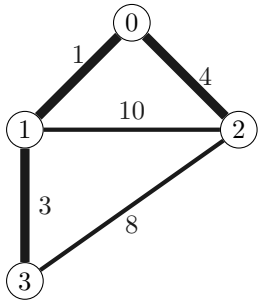
where $\mathrm{dist}_T(r_o, r_d)$ is the cost of the path from r_o to r_d in T.

A spanning tree in G that minimizes (3.1) is called an **optimal communication spanning tree**. In the OCST problem, for a given instance $\mathcal{I} = (G, c, R, w)$, the objective is to

find an optimal communication spanning tree.

For a spanning tree T and a requirement $r \in R$, the **requirement path** associated with r, denoted by P_{r_o,r_d}, is the path in T that links r_o to r_d. Observe that, by convention we consider that the first vertex of the pair is the origin and the second is the destination; for the cost of the corresponding requirement path it does not matter (as it is the distance between the origin and destination), but the pairs (u, v) and (u, v) may have different demands, and both pairs may be in R. We say that R is **complete** if, for each pair of vertices $u, v \in V$, either (u, v) or (v, u) belongs to R.

Example 3.1. An instance of the OCST problem is illustrated in Figure 3.1, together with a feasible solution T, indicated with ticker edges. To determine the communication cost of T, we calculate the cost of P_{r_o,r_d}, for each requirement $r \in R$. The requirement $(0, 1)$ has demand 4, and $P_{0,1}$ is composed by the edge $\{0, 1\}$ whose cost is 1. Thus, it contributes with cost $w_{0,1}\, c_{0,1} = 4 \cdot 1 = 4$ to the total communication cost. In a similar way, requirement $(1, 2)$ contributes with cost $10 = 2(1+4)$, and requirement $(2, 3)$ contributes with cost $8 = 4+1+3$. Therefore, the communication cost of T is 22.



Requirement (r)	**Demand** (w_r)
$(0, 1)$	4
$(1, 2)$	2
$(2, 3)$	1

Figure 3.1: *An instance of the OCST problem and a feasible solution T, indicated with ticker edges, which is also an optimal communication tree.*

3.2 Historical review

Spanning trees arise very often in many combinatorial optimization problems. We recall that a spanning tree of a connected graph G is a connected acyclic subgraph of G that spans $V(G)$. One of the most studied problems involving spanning trees is the *Minimum Spanning Tree* (MST) problem, defined as follows: given a connected graph G with costs associated with its edges, find in G a spanning tree of minimum total cost.

An algorithm for the MST problem was first presented in 1926 by Borůvka [NN12]. In the fifties other polynomial-time algorithms to solve it were proposed, being the greedy algorithm proposed by Kruskal [Kru56] the best known. However, many optimization problems in which a spanning tree with certain properties are required, are known to become hard. This is the

case of the OCST problem. In 1978, Johnson, Lenstra, and Kan [JLK78] proved that OCST is NP-hard, even if all the requirements have the same demand.

In view of this, many approaches have been proposed to solve special cases of OCST, as well as heuristics [HLN10, FM07, NLL13] and approximation algorithms [WC04]. Exact algorithms [AM87] and MILP formulations have also been proposed [FLH+13, CFM09, Lun16, ZCFL19, TI18].

In what follows, we present a historical review of the main results that appeared in the literature regarding this problem. For every period of time, we mention the most relevant results.

3.2.1 From 1974 to 1984: introduction of the problem

The OCST problem was introduced in 1974 by T. C. Hu [Hu.74]. He studied two special cases of OCST: the Optimum Requirement Spanning Tree (ORST) problem, and a particular case of the Minimum Routing Cost Spanning Tree (MRCT) problem.

In 1984, Agarwal and Venkateshan [AMS84] presented polynomial-time algorithms for two constrained cases of ORST. In the first case, some vertices are required to be leaves of the solution. In the second case, certain pairs of vertices are required to be linked by an edge in the solution. Finally, he also studied how the structure of a solution for an instance of ORST changes when we modify the demand of a requirement.

3.2.2 From 1987 to 2012: heuristics and genetic algorithms

Ahuja and Marty [AM87] developed a branch-and-bound approach for OCST and solved instances with up to 40 vertices. They also proposed a heuristic algorithm that finds near-optimal solutions for graphs with 100 vertices. This algorithm consists of two phases. In the first phase, it finds a spanning tree T in G; then, it tries to find another tree with better communication cost by exchanging one edge in $E(T)$ with one in $E(G) \setminus E(T)$. In 1994, Palmer and Kershenbaum [PK94] proposed two approaches for OCST: a heuristic algorithm and a genetic algorithm. Later, in 1995, these authors [PK95] and then Soak [Soa06] in 2006, proposed genetic algorithms for OCST. In 2011, Hieu, Quoc and Nghia [NQN11] designed an *ant colony* heuristics for MRCT. Later, in 2012, Tan [Tan12] presented a genetic approach for the same problem.

3.2.3 From 1980 on: approximation algorithms

Let (G, c, R, w) be an instance of OCST, where $G = (V, E)$. Let $\phi : V \to \mathbb{R}_{\geq 0}$, and let $S = \{r_o : r \in R\}$ be the set of origins of the pairs in R. In what follows, we define variants of the OCST problem that have been studied in the literature. The additional hypotheses on the instances of OCST (besides ϕ and S defined above) are stated for each variant.

- Minimum Routing Cost Spanning Tree (MRCT): R is complete, and $w_r = 1$, $\forall r \in R$.
- Optimal Product-Requirement Communication Spanning Tree (PROCT): R is complete, and $w_r = \phi(r_o) \cdot \phi(r_d)$, $\forall r \in R$.
- Optimal Sum-Requirement Communication Spanning Tree (SROCT): R is complete, and $w_r = \phi(r_o) + \phi(r_d)$, $\forall r \in R$.
- p-Source OCST (p-OCST): $|S| = p$
- p-Source MRCT (p-MRCT): $|S| = p$; $w_r = 1, \forall r \in R$; and either (u, v) or (v, u) belongs to R, $\forall u \in S$ and $\forall v \in V$.

The relationship between these problems, according to Wu and Chao [WC04], is shown in Figure 3.2.

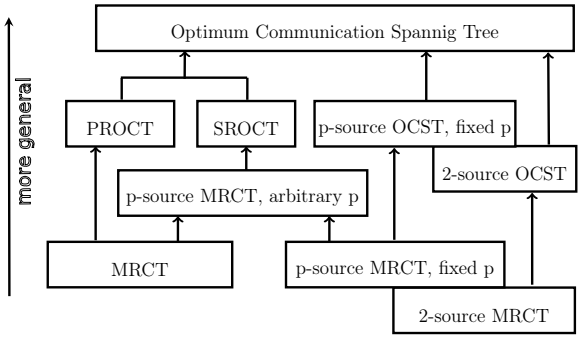


Figure 3.2: *Relationship between variants of* OCST.

Johnson et al. [JLK78] showed that MRCT is NP-hard. Since MRCT is a particular case of PROCT and SROCT, these problems are also NP-hard. In 1980, Wong [Won80] proposed a 2-approximation algorithm for MRTC. His algorithm chooses a shortest path tree that minimizes the overall sum of distances from the root to each vertex. In 1996, Bartal [Bar96] proposed a probabilistic approximation algorithm for OCST. Later, Bartal [Bar98] proposed an approximation algorithm for the metric case of the OCST with factor $\log n \log \log n$. In the same year, Peleg and Reshef [PR98] presented approximation algorithms for several variants of OCST. Moreover, they showed that OCST is MAX SNP-hard. Thus, it does not admit a PTAS, unless P = NP.

In 2000, Wu et al. [WCT00b, WLB+00, WCT00a, WCT00c] obtained several results regarding MRCT, PROCT and SROCT. In [WCT00b], they proposed some approximation algorithms for MRCT. In [WLB+00], they showed that MRCT is NP-hard, even if the edge-costs obey the triangle inequality. Moreover, they designed a $O(|V|^3)$ algorithm that transforms an instance of MRCT into a metric instance (metric case). In [WCT00a], they showed a similar result for PROCT. Futhermore, they proposed a 1.577-approximation algorithm for PROCT, and a 2-approximation for SROCT. Finally, in [WCT00c], they proposed a PTAS for PROCT.

In 2002, Wu [Wu02] showed that the 2-MRCT is NP-hard, and proposed a PTAS for it. Wu and Chao [Wu04], in 2004, showed that the p-OCST is NP-hard, for $p \geq 2$. For the

metric case, on graphs of order n, they presented a 2-approximation algorithm with time complexity $O(n^{p-1})$. Moreover, they showed a 3-approximation algorithm for the general case.

In 2015, Ravelo and Ferreira [RF15] presented a PTAS for the metric case of SROCT. Later in 2017, they studied special cases of the metric p-OCST [RF17]. The previously mentioned results are listed in Table 3.1.

Problem	Authors	Approx. factor	Running time
MRCT	Wu et.al. [WLB+00]	$(1+\epsilon)$	$n^{O(1/\epsilon)}$
	Wu, Chao and Tang [WCT00a]	1.577	$O(n^3)$
PROCT	Wu, Chao and Tang [WCT00c, WCT00a]	$(k+3)/(k+1)$	$n^{(k-1)}$
		1.577	$O(n^5)$
SROCT	Wu, Chao and Tang [WCT00a]	2	$O(n^3)$
metric SROCT	Ravelo and Ferreira [RF17]	$1+\frac{\epsilon}{1-\epsilon}$	$n^{6\left(\lceil\frac{6}{\epsilon}\rceil^2-11\lceil\frac{6}{\epsilon}\rceil+1\right)+1}\log^2(n)$
p-MRCT	Ravelo and Ferreira [RF15]	$(1+\epsilon)$	$n^{\left(\frac{8p^2}{\epsilon(2p-\epsilon)}-1\right)(p-1)+1}$
p-OCST	Wu [Wu04]	2	$O(n^{p-1})$
2-MRCT	Wu [Wu02]	$(1+\epsilon)$	$O(n^{\frac{1}{\epsilon}+1})$

Table 3.1: *Approximation algorithms.*

3.2.4 From 2002 on: MILP formulations, large-scale optimization

In 2002, Fischetti, Lancia, and Serafini [FLS02] presented two formulations for MRCT. Regarding OCST, in 2010, Contreras et al. [CFM10a] showed lower bounds for OCST based on a Lagrangian relaxation. Fernández, Luna, Hildenbrandt, Reinelt, and Wiesberg [FLH+13] proposed a flow formulation in 2013; and in 2015, Luna [Lun16] proposed a formulation based on rooted trees.

In 2018, Tilk and Irnich [TI18] proposed branch-and-cut-and-price algorithms based on Dantzig-Wolfe decomposition. These algorithms obtain optimal solutions for instances of OCST with up to 40 vertices and edge density above 35%.

In 2019, Zetina, Contreras, Fernández and Luna [ZCFL19] proposed an algorithm based on Benders decomposition, integrated within a branch-and-bound framework. Computational experiments with this algorithm show that it outperforms the algorithms proposed by Tilk and Irnich [TI18], expanding the limits of solvability for the OCST problem from 40 to 60 vertices. In terms of exact algorithms, this is the algorithm with best performance.

Agarwal and Venkateshan [AV19] proposed new valid inequalities (and separation routines for them) for a flow-based formulation to solve the OCST problem. These authors show –on small instances– computational evidence of the strength of these new inequalities, but they do not show an algorithm that outperforms the algorithm we have previously mentioned.

3.3 Applications

The OCST problem and its variants have many applications. One of them is on the Multiple Sequence Alignment (MSA) problem [WLB+00]. In this problem, given a set of n strings, our objective is to insert gaps into these strings in such a way that: all the strings have the same length, and when we consider a position, they all contain the same character (a gap match every character). The goal is to minimize the total number of gaps.

An example for the strings ATTCG, TTCCG and ATCG is presented in Figure 3.3. A gap is represented by the character '-'.

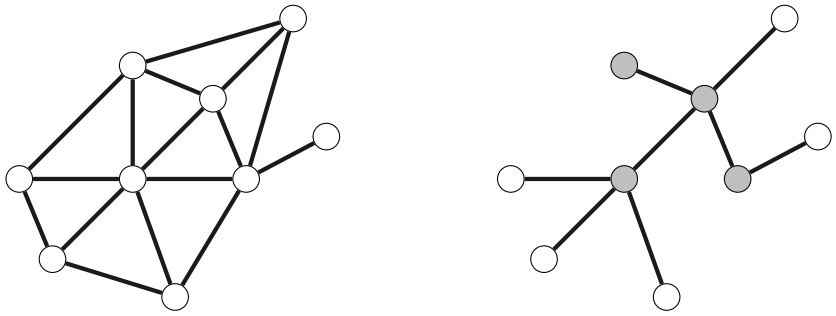
A	T	T	C	-	G
-	T	T	C	C	G
A	-	T	C	-	G

Figure 3.3: *An alignment for strings s_1= ATTCG, s_2=TTCCG and s_3=ATCG.*

Waterman [Wat95] designed a polynomial-time algorithm for MSA when we consider instances with two strings. If we consider instances with at least three strings, the decision version of this problem is NP-complete [WJ94]. For the general case, an exponential algorithm, using dynamic programming, was proposed by Sankoff [SK83]. This algorithm has complexity $O(2^n \ell^n)$, where ℓ is the maximum length of the n strings.

Most of the approaches used to solve MSA construct an alignment sequentially by merging two strings, and by adding the alignment of both strings, until an alignment for all of them is obtained. Thus, the order in which the merge of the strings is carried out affects the total cost of the final alignment. The advantage of this approach is its low running time [WLB+00]. In order to find an upper bound for the total cost of this alignment, we model it as an instance of OCST, as follows. In the current stage of the procedure, we consider a graph, say G', such that every string is represented by a vertex in G'. Moreover, every pair of vertices is linked by an edge. For each edge, its cost is equal to the edit distance (the cost of turning one string into the other) of its ends. Finally, we set the demand for each requirement to 1. Since OCST is NP-hard, approximate solutions are used to maintain a reasonable running time.

Another application of OCST occurs in the Tree-of-Hubs Location (THL) problem [CFM10b]. An instance of THL is given by a tuple (G, c, R, w, p, α), where (G, c, R, w) is defined in an analogous way as in OCST. Besides that, p is a positive integer and α is a positive real number such that $p \leq |V(G)|$ and $\alpha \leq 1$. Moreover, p and α are related to the size of a feasible solution and the objective value, respectively. In this problem, we seek two trees in G, say T and T', such that T is a spanning tree of G, and T' is a subtree of T of order p. Moreover, each edge of T must have at least one end in $V(T')$. In other words, each vertex $v \in V(T) \setminus V(T')$ is a leaf in T. In Figure 3.4, we show an instance of THL with $p = 4$, and a feasible solution (T, T') where the vertices in gray belong to T'.



(a) (b)

Figure 3.4: (a) *A graph G of an instance $\mathcal{I} = (G, c, R, w, p, \alpha)$, where $p = 4$;* (b) *a feasible solution (T, T') for $\mathcal{I}$, where the gray vertices belong to T'.*

In THL, the vertices of T' are called **hubs**. The edges that link hubs represent "fast" connections. So, they receive a "discount" in their cost that is determined by α. Thus, to define the objective function, we use an auxiliary cost function c', associated with $E(T)$, defined as follows.

$$c'_{ij} := \begin{cases} \alpha c_{ij}, & \text{if } ij \in E(T') \\ c_{ij}, & \text{otherwise.} \end{cases} \tag{3.2}$$

Hence, the objective function that we want to minimize is the following:

$$\sum_{r \in R} w_r \operatorname{dist}_T(r_o, r_d), \tag{3.3}$$

where $\operatorname{dist}_T(r_o, r_d)$ is the cost, considering c', of the path between r_o and r_d in T. In what follows, we describe a special case of THL, and show how we can model it as an OCST instance. Let $\mathcal{I} = (G, c, R, w, p, \alpha)$ be an instance of THL. Let $V' \subseteq V$ be such that $|V'| = p$ and $|N_G(v) \cap V'| = 1$, for each $v \in V(G) \setminus V'$. We are interested in finding a THL such that $V(T') = V'$. That is, we want to find an optimal tree such that the hub nodes are precisely the set V'. Now, we show how to reduce this problem to OCST. Let h be a function associated with vertices of G such that $h(u)$ is the hub linked to u, if $u \in V(G) \setminus V'$. Otherwise $h(u) = u$. Let (T, T') be a feasible solution, and let $r \in R$ be a requirement such that $u = r_o$ and $v = r_d$. Let P be the path from u to v in T. Note that P is composed of three subpaths, say P_1, P_2 and P_3, such that

- P_1 is a path from u to $h(u)$ (possibly empty),
- P_2 is a path from $h(u)$ to $h(v)$ in T', and
- P_3 is a path from $h(v)$ to v (possibly empty).

Thus, we have that

$$\begin{aligned}\sum_{(u,v)\in R} w_r \mathrm{dist}_T(u,v) &= \sum_{(u,v)\in R} w_r \left(c_{u,h(u)} + \alpha\, \mathrm{dist}_{T'}(h(u),h(v)) + c_{v,h(v)}\right) \\ &= \sum_{(u,v)\in R} w_r \left(c_{u,h(u)} + c_{v,h(v)}\right) + \alpha \sum_{(u,v)\in R} w_r\, \mathrm{dist}_{T'}(h(u),h(v)).\end{aligned}$$

Since $|N_G(v) \cap V'| = 1$, for $v \in V(G) \setminus V'$, the first summation in the right side of the expression above is a constant. As $\alpha \geq 0$, the problem reduces to finding an OCST of the subgraph induced by V'.

A real application of the previous problem occurred in the design of a high-speed train network in Spain [CFM10b]. In this case, the goal was to construct train stations in some fixed cities — those with higher population (the set V'). Due to the high cost for building the network, it was required to have a tree topology. Moreover, cities without a station (vertices not belonging to V') were required to be connected to a near city with a station. Furthermore, the average number of people that would travel between cities was known (the demand). Thus, this problem could be modelled as a special case of THL.

3.4 Special cases of the OCST problem

In this section, we present special cases of the OCST problem that can be solved in polynomial time. First, we consider the 1-source OCST problem. After that, we present two special cases that were investigated by Hu [Hu.74]. In both cases, the input graph is complete.

3.4.1 1-source MRCT

Given an instance of MRCT, if the origin of all the requirements is the same vertex, say o, the objective function is the following:

$$\sum_{r\in R} \mathrm{dist}_T(o, r_d). \tag{3.4}$$

Moreover, when $|R| = 1$, we are are looking for a shortest path between two vertices. For $|R| > 1$, the tree we are looking for is precisely a shortest path tree with root o [Wu02]. The latter problem has been extensively studied. Many polynomial-time algorithms have been proposed for solving it, as for example, Dijkstra's algorithm [Dij59].

3.4.2 Optimum Requirement Spanning Tree

The Optimum Requirement Spanning Tree (ORST) problem is a particular case of OCST in which the input graph is complete and each edge has a cost equal to 1 ($c : E \to \{1\}$). Thus, in this case, we want to find a spanning tree T that minimizes:

$$\sum_{(r_o,r_d)\in R} w_r |P_{r_o,r_d}|, \tag{3.5}$$

where P_{r_o,r_d} denotes the path between r_o and r_d in T.

In what follows, we show how to reduce ORST to a multiterminal network flow problem and how to associate with it a tree, defined as the Gomory-Hu tree [GH61] of this network. Before, we present the concept of a laminar family consisting of subsets of the vertex set of the input graph G. This concept is important in the construction of the Gomory-Hu tree.

Laminar families

Let N be a finite set and let A, B be subsets of N. We say that A and B **cross** or is a **crossing pair** if

$$A \cap B \neq \emptyset, \quad A \not\subseteq B, \quad B \not\subseteq A, \quad \text{and} \quad A \cup B \neq N.$$

Similarly, we say that A and B **overlap** if

$$A \cap B \neq \emptyset, \quad A \not\subseteq B, \quad \text{and} \quad B \not\subseteq A.$$

A family L of sets of a finite set is **cross free** (resp. **overlap free**) if no two sets $A, B \in L$ cross (resp. overlap).

Overlap free families are better known as **laminar families**, the terminology that we will adopt. Clearly, a laminar family is also a cross free family. Moreover, a laminar family, over a ground set N, has at most $2|N| - 1$ distinct elements [BHR12].

The Gomory-Hu tree

Given the network $N = (G, u)$, where $G = (V, E)$, the Multiterminal Network Flow (MNF) problem consists in finding a min-cut for every pair of vertices in G. Gomory and Hu [GH61] studied this problem and showed a polynomial-time algorithm that receives, as input, the network N and returns, as output, a capacitated tree. This tree is called the **Gomory-Hu tree** of N, and its properties are stated in the following theorem.

Theorem 3.4.1 (Gomory and Hu, 1961)**.** *Let (G, u) be a network, (T, c) be a Gomory-Hu tree of (G, u). Let s and t be two distinct vertices in G. The network (T, c) satisfies the following properties:*

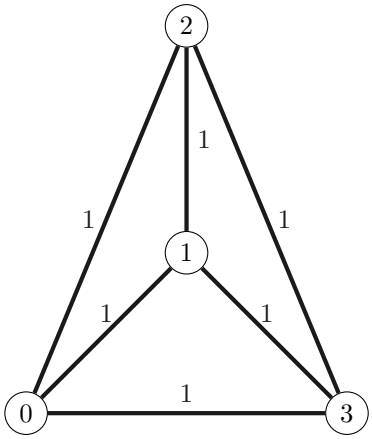
1. *Let P_{st} be the path linking s and t in T. The capacity of a minimum s-t cut in G is equal to $\min\{c_e : e \in E(P_{st})\}$.*
2. *If the edge e^* achieves the above minimum, then a minimum s-t cut, in G, is given by the bipartition of V corresponding to the vertices in the two components of $T - e^*$.*

Polynomial-time algorithm for the ORST problem

In this subsection we show that ORST can be solved in polynomial time using the Gomory-Hu tree of a certain network [GH61]. Let $\mathcal{I} = (G, c, R, w)$ be an instance of ORST, where $G = (V, E)$ is a complete graph of order n, and $c_e = 1$, for every $e \in E$. Let $u : E \to \mathbb{R}$ be a capacity function defined as $u_{ij} = w_{ij} + w_{ji}$, for $i, j \in V, i \neq j$. Without loss of generality, we assume that w_r is equal to zero if $r \notin R$. Throughout this subsection, we say that (G, u) is the **associated** network of $\mathcal{I}$.

Hu [Hu.74] showed that the sum of capacities of a Gomory-Hu tree of (G, u) is equal to the optimal communication cost of $\mathcal{I}$. In Figure 3.5, we show an instance $\mathcal{I} = (G, c, R, w)$ of ORST, and in Figure 3.6 we show its associated network (G, u).

Given a graph $G = (V, E)$, laminar families over V play an important role in the algorithm for ORST proposed by Hu. In particular, we will consider that each set X in a laminar family L represents a cut in G. In what follows, we show that, in order to contain a cut that separates every pair of vertices in G, we must have that $|L| \geq |V| - 1$.



Requirement (r)	**Demand** (w_r)
$(0, 1)$	4
$(0, 2)$	3
$(0, 3)$	2
$(1, 2)$	7
$(1, 3)$	3
$(2, 3)$	6
$(3, 2)$	2

Figure 3.5: *An instance $\mathcal{I}$ of the* ORST *problem.*

Lemma 3.4.2 (Hu, 1974)**.** *Let $G = (V, E)$ be a graph of order n, and let L be a laminar family over V. If $|L| < n - 1$, then there exist vertices, say s and t, such that either $s, t \in Z$ or $s, t \in V \setminus Z$, for each $Z \in L$.*

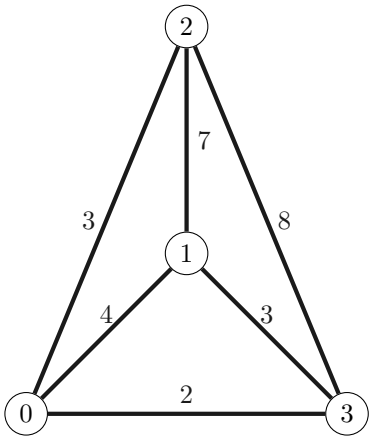


Figure 3.6: *The associated network (G, u) of the instance $\mathcal{I}$, where $u_{ij} = w_{ij} + w_{ji}$.*

Proof. We show this by induction on n. If $n = 2$, then $|L| = 0$. So, the condition holds. Now, suppose that $n \geq 3$. Let X be a maximal set of L, and let $Y = V \setminus X$. Consider the laminar families $L^X = \{Z : Z \subset X, Z \in L\}$ and $L^Y = \{Z : Z \subseteq Y, Z \in L\}$. Also, consider the graphs $G^X = G[X]$ and $G^Y = G[Y]$. If $|L^X| < |X| - 1$ or $|L^Y| < |Y| - 1$, by the induction hypothesis the result follows. Thus, suppose that $|L^X| \geq |X| - 1$ and $|L^Y| \geq |Y| - 1$. Since $L = L^X \cup L^Y \cup \{X\}$, we have that

$$|L| \geq |L^X| + |L^Y| + 1 \geq |X| - 1 + |Y| - 1 + 1 = n - 1,$$

a contradiction. □

Let $G = (V, E)$ be a connected graph of order n, and let L be a laminar family over V such that $|L| = n - 1$. We say that L is a **separating family** if, for every pair of vertices i and j in G, there is a set $X \in L$ such that $\delta(X)$ is an i-j cut. Now, let T be a spanning tree of G, and let L be a separating family. We say that L **corresponds** to T (and also T **corresponds** to L) if, for every $X \in L$, we have that $T[X]$ is connected. The following result relates the spanning trees of a graph G to separating families over $V(G)$.

Lemma 3.4.3 (Hu, 1974)**.** *Let $G = (V, E)$ be a graph of order n. Let T be a spanning tree of G, and let L be a laminar family of size $n - 1$. Then, the following hold:*

1. *there is a separating family L' that corresponds to T, and*
2. *there is a tree T' that corresponds to L.*

Proof. We will show both claims by induction on n. First, we show statement 1. If $n = 1$, then T is a single vertex. Thus, $L' = \emptyset$. So, suppose that $n \geq 2$. Let $e \in E(T)$. Let T^1 and T^2 be the components (trees) of $T - e$. By induction hypothesis, there exist separating families, say L^1 and L^2, that correspond to T^1 and T^2, respectively. Observe that $L' = L^1 \cup L^2 \cup \{V(T^1)\}$ is a separating family that corresponds to T.

Now, we show statement 2. If $n = 1$, then a tree T' with a single vertex corresponds to L.

Suppose that $n \geq 2$. Let X be a maximal set of L, distinct from V, and let $Y = V \setminus X$. Consider the laminar families $L^X = \{Z : Z \subset X, Z \in L\}$ and $L^Y = \{Z : Z \subset Y, Z \in L\}$. Observe that $|L^X| = |X| - 1$ and $|L^Y| = |Y| - 1$. Otherwise, $|L^X| < |X| - 1$ or $|L^Y| < |Y| - 1$ and, by Lemma 3.4.2, there would exist a pair of vertices in T not separated by L. Thus, by induction hypothesis, there exist trees, say T^X and T^Y, that correspond to L^X and L^Y, respectively. Let x (resp. y) be a vertex in T^X (resp. T^Y) such that xy is an edge in G (it exists because G is connected). Therefore, L corresponds to $T' = T^X \cup T^Y \cup \{xy\}$. □

In Figure 3.7, we show two examples of separating families that correspond to two spanning trees. Nodes in gray belong to the laminar cuts associated with each one of the former trees. Let T be a tree, and let L be a separating family that corresponds to T. The next two results will relate the number of cuts (induced by sets in L) that separate two vertices to its distance in T.

Proposition 3.4.4 (Hu, 1974)**.** *Let $G = (V, E)$ be a connected graph, T a spanning tree of G, and L a separating family that corresponds to T. There is a unique set $X \in L$ such that $\delta(X)$ is an i-j cut, for each edge $ij \in E(T)$.*

Proof. We show the claim by induction on $n := |V|$. If $n = 1$, the claim trivialy follows. So, suppose that $n \geq 2$. Let $ij \in E(T)$. We consider two cases.

Case 1: The vertex i or the vertex j is a leaf of T.

Without loss of generality, suppose that i is a leaf of T. Let $X \in L$ induce a cut that separates i from j. Since i is a leaf either $X = \{i\}$ or $X = V \setminus \{i\}$. Suppose that $X = \{i\}$ and $Y = V \setminus \{i\}$. Moreover, as $|L| = n - 1$, L contains just one of these sets. Otherwise it would contradict Lemma 3.4.2, since $L' = L \setminus \{X, Y\}$ is a separating family of $T - i$ with size $n - 2$.

Case 2: The vertices i and j are internal vertices of T.

Since $n \geq 2$, T must contain a leaf, say k. Let p be the neighbor of k in T, and let $X \in L$ induce a p-k cut. By the previous case, such set X is unique. Now, consider the family $L' = L \setminus \{X\}$, and the tree $T' = T - k$. Observe that L' is a separating family that corresponds to T'. Since $ij \in E(T')$, by induction hypothesis, there is a unique set $X' \in L'$ that induces an i-j cut. As $\delta(X)$ is not an i-j cut, and $X' \in L$, the claim follows. □

Corollary 3.4.5. *Let $G = (V, E)$, T and L be as stated in Proposition 3.4.4. For each edge $e \in E(T)$, denote by X_e the set in L that induces a cut in T that separates its ends. Consider two distinct vertices i and j in T, and the set S_{ij} defined as follows:*

$$S_{ij} = \{X \in L : \ \delta(X) \text{ is an } i\text{-}j \text{ cut}\}.$$

Then,

$$S_{ij} = \{X_e : e \in P_{ij}\},$$

where P_{ij} is the path in T that links i to j.

Proof. First, observe that for every set X that induces an i-j cut, it must contain a unique edge in $E(P_{ij})$ since $T[X]$ is connected. Thus, by Proposition 3.4.4, the result follows. □

Consider an instance $\mathcal{I} = (G, c, R, w)$ of ORST. Note that since G is complete, any tree with the same vertex set is a spanning tree of G. In what follows, we show that the communication cost of a feasible solution of $\mathcal{I}$, say T, is equal to the sum of the capacities of a separating family that corresponds to T.

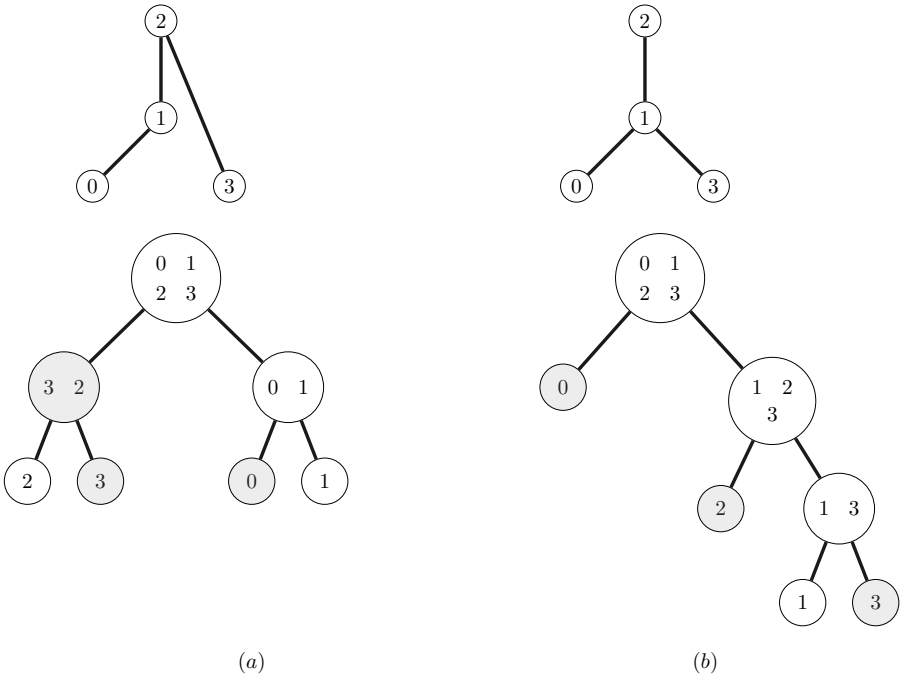


Figure 3.7: (*a*) *Cuts (in gray) correspond to edges* $\{1,2\}$, $\{2,3\}$ *and* $\{0,1\}$*;* (*b*) *cuts (in gray) correspond to edges* $\{0,1\}$, $\{1,2\}$ *and* $\{1,3\}$.

Lemma 3.4.6 (Hu, 1974)**.** *Let $\mathcal{I} = (G, c, R, w)$ be an instance of* ORST*, and let T be a spanning tree of G. The communication cost of T is equal to*

$$\sum_{X \in L} u(\delta(X)),$$

where L is a separating family that corresponds to T, and (G, u) is the network associated with $\mathcal{I}$.

Proof. Let $X \in L$. Observe that $u(\delta(X)) = \sum_{s \in X} \sum_{t \in V \setminus X} u_{st}$. So, the sum of capacities of

all the sets in L is equal to:

$$\sum_{X \in L} u(\delta(X)) = \sum_{X \in L} \sum_{s \in X} \sum_{t \in V \setminus X} u_{st}. \tag{3.6}$$

By Corollary 3.4.5 and (3.6), we have that

$$\begin{aligned} \sum_{S \in L} u(\delta(S)) &= \sum_{ij \in E(G)} u_{ij} \text{dist}_T(i,j), \\ &= \sum_{ij \in E(G)} (w_{ij} + w_{ji}) \text{dist}_T(i,j). \end{aligned} \tag{3.7}$$

□

In what follows, $\mathcal{I} = (G, c, R, w)$ is an instance of ORST, and (G, u) is its associated network. Let T be a spanning tree of G, and L a separating family that corresponds to T. Since $|E(T)| = |L|$, Propositon 3.4.4 and Lemma 3.4.6 imply that the communication cost of T is equal to

$$\sum_{e \in E(T)} u(\delta(X_e)), \tag{3.8}$$

where $X_e \in L$ is the set that induces a cut that separates the ends of e. Adolphson and Hu [AH73] showed that, given two spanning trees over the same vertex set, there is a bijection between its edges that satisfy an important property.

Lemma 3.4.7 (Adolphson and Hu, 1973)**.** *Let $G = (V, E)$ be a graph and let B and R be spanning trees of G. Then, there exists a bijection $\varphi : E(B) \to E(R)$ such that $\varphi(ij) \in E(P_{ij})$, for every $ij \in E(B)$, where P_{ij} is the path that links i and j in R.*

Proof. Let G' be a (X, Y)-bipartite graph such that $X = E(B)$ and $Y = E(R)$. Moreover, $N_{G'}(ij) = E(P_{ij})$, for every $ij \in E(B)$. That is, an edge $ij \in E(B)$ is linked to the edges in the path between i and j in R. We show an example of this construction in Figure 3.8. Observe that the existence of φ is equivalent to showing that G' has a perfect matching. To show that, we will use Hall's Theorem.

Let E_B be a subset of edges in $E(B)$. We will show that

$$|E_B| \leq |N_{G'}(E_B)|.$$

Let B' (resp. R') be the subgraph spanned by E_B (resp. $N_{G'}(E_B)$) in B (resp. R). Moreover, let us denote by n_B and k_B (resp. n_R and k_R) the number of vertices and components, respectively, of B' (resp. R'). First, if $i \in V(B')$, there is an edge $e \in E_B$ incident to i. By definition, there exists an edge in $N_{G'}(e)$ incident to i. Thus, $i \in V(R')$. Therefore, $n_B \leq n_R$. Now, since $N_{G'}(e)$ induces a path in R, for every $e \in E(B)$, the vertices of a component

in B' are contained in a component of R' which implies that $k_R \leq k_B$. Therefore, we have that

$$|E_B| = n_B - k_B \leq n_R - k_R = |N_{G'}(E_B)|.$$

□

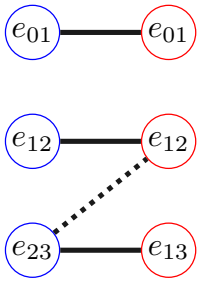


Figure 3.8: *The bipartite graph G' obtained from the two trees in Figure 3.7.*

Finally, we present the main result of this subsection.

Theorem 3.4.8 (Adolphson and Hu, 1973)**.** *Let $\mathcal{I} = (G, c, R, w)$ be an instance of* ORST *problem, and let T be a Gomory-Hu tree of the network (G, u). Then, T is an optimal solution for the instance $\mathcal{I}$.*

Proof. Let T^* be an optimal solution for the instance $\mathcal{I}$. Let L and L^* be separating families that correspond to T and T^*, respectively. By Lemma 3.4.6, it suffices to show that

$$\sum_{e \in E(T)} u(\delta(X_e)) \leq \sum_{f \in E(T^*)} u(\delta(X_f^*)).$$

Let φ be a bijection between $E(T)$ and $E(T^*)$ as in Lemma 3.4.7. Now, let $e = ij$ be an edge in T, and let $f = \varphi(e)$ be an edge in T^*. By Theorem 3.4.1, X_e is a minimum i-j cut in G. On the other hand, since f belongs to the path between i and j in T^*, we have that $X_f^* \in L^*$ is also an i-j cut. Thus, $u(\delta(X_e)) \leq u(\delta(X_f^*))$. Therefore, since φ is a bijection, adding up this inequality for each edge $e \in E(T)$, we obtain

$$\sum_{e \in E(T)} u(\delta(X_e)) \leq \sum_{f \in E(T^*)} u(\delta(X_f^*)).$$

□

A Gomory–Hu tree of an n-vertex network can be constructed by considering at most $n - 1$ maximum-flow min-cut computations. Many polynomial-time algorithms have been proposed to construct it, as for example, Push-relabel with FIFO vertex selection rule algorithm [GT88], Dinitz's algorithm [Din06], Ford-Fulkerson algorithm [FF57], among others.

3.4.3 A special case of the Minimum Routing Cost Spanning Tree

The Minimum Routing Cost Spanning Tree (MRCT) is a particular case of OCST in which each requirement has demand equal to 1, and R is complete. The MRCT problem is also called the optimum distance spanning tree problem [Hu.74]. The goal is to find a spanning tree T that minimizes the following objective function:

$$\sum_{r \in R} \text{dist}_T(r_o, r_d). \tag{3.9}$$

As we mentioned before, Johnson et al. [JLK78] showed that the MRCT problem is NP-hard.

We present here two special cases of MRCT that can be solved in polynomial time. These results were obtained by Hu [Hu.74]. The first is for MRCT, restricted to instances with constant cost function c; the other is a slightly stronger result in which the cost function c need not to be constant, but must satisfy a certain condition.

Let (G, c, R, w) be an instance of MRCT where G is a complete graph of order n. Let T be spanning tree of G. Let e be any edge in T and let T_1 and T_2 the components of $T - e$. Clearly, for any $u \in V(T_1)$ and $v \in V(T_2)$, the path that links u and v, in T, contains the edge e. Moreover, no path that links two vertices in $V(T_1)$ or in $V(T_2)$ contains e. Therefore, the contribution to the communication cost that corresponds to the edge e is $c_e \cdot k \cdot (n-k)$, where $k = |V(T_1)|$. So, we can calculate the communication cost of T by adding up the contribution of each edge in T.

In Lemma 3.4.9, we show that if all edges have the same cost, then for any instance of MRCT, an optimal solution is a star. Let T be a tree. We recall that an internal vertex of T is a vertex that is not a leaf, and that an internal edge is an edge that links two internal vertices. Moreover, an extremal internal vertex of T is an internal vertex that is adjacent to a leaf.

Lemma 3.4.9 (Hu, 1974)**.** *Let $\mathcal{I} = (G, c, R, w)$ be an instance of the* MRCT *problem, where G is a complete graph of order n. If all the edges of G have the same cost, then any optimal solution for $\mathcal{I}$ is a star.*

Proof. The proof is by contradiction. Note that if $n \leq 3$, any spanning tree of G is a star. So, we can suppose that $n \geq 4$. Suppose that there exists an optimal solution T^* of $\mathcal{I}$ that is not a star. Let c^* be the cost of every edge of G. We will consider the communtication cost of T^* as the sum of a) the communtication cost of every internal edge of T, and b) the commutication cost of every edge incident to a leaf.

First, observe that for every edge incident to a leaf, its communication cost is $c^*(n-1)$. Let e be an internal edge of T^*. Moreover, let k be the order of one of the components

in $T^* - e$. Thus, the communication cost of e is $c^*k(n-k)$, where $k \geq 2$ and $n - k \geq 2$.

Observe that $c^*k(n-k) > c^*(n-1)$ if $n \geq 4$. Consider an edge $e = uv$ such that v is an extremal interval vertex. Then, the tree T' obtained from T^* by removing every edge incident to v, and by linking every isolated vertex to u, has less communication cost than T^*, a contradiction. Therefore, in an optimal solution for $\mathcal{I}$, every edge must be incident to a leaf. □

The next result, for a slightly more general cost function c, also ensures the existence of an optimal solution that is a star.

Theorem 3.4.10 (Hu, 1974)**.** *Let $\mathcal{I} = (G, c, R, w)$ be an instance of the* MRCT *problem, where $G = (V, E)$ is a complete graph of order $n \geq 4$. Suppose that there exists a nonnegative number $t \leq (n-2)/(2n-2)$ such that*

$$c_{ij} + tc_{jk} \geq c_{ik}, \quad \forall i, j, k \in V.$$

Then, there exists a star that is an optimal solution for $\mathcal{I}$.

Proof. Suppose by contradiction that such optimal solution does not exist. Among the optimal solutions, let T^* be an optimal solution with the maximum number of leaves possible.

Let q be an extremal internal vertex of T^* adjacent to the least number of leaves. Moreover, let p be an internal vertex adjacent to q in T^*. Let $v_1, \ldots, v_{k-1}$ be the leaves adjacent to q in T^*. Since T^* contains at least two internal vertices, we have that $k \leq n/2$. Now, let T be a spanning tree of G such that

$$E(T) = E(T^*) \setminus \{qu : u \in V, uq \in E(T^*), u \neq p\} \cup \{pu : u \in V, uq \in E(T^*), u \neq p\}.$$

Observe that T has one more leaf than T^*. We will show that T has optimal communication cost, contradicting the choice of T^*. In Figure 3.9, we show how T is constructed from T^*.

Observe that the contribution to the communication cost of T^* of the edges $\{pq\} \cup E(T^*) \setminus E(T)$ is:

$$c_{pq}k(n-k) + \sum_{i=1}^{k-1} c_{qv_i}(n-1). \tag{3.10}$$

On the other hand, the contribution of the edges in $\{pq\} \cup E(T) \setminus E(T^*)$ to the communication cost of T is:

$$c_{pq}(n-1) + \sum_{i=1}^{k-1} c_{pv_i}(n-1). \tag{3.11}$$

Thus, the difference between the communication cost of T^* and T is

$$(n-1)(c_{pq})\left[\frac{k(n-k)}{n-1}-1\right]+(n-1)\left[\sum_{i=1}^{k-1} c_{qv_i}-c_{pv_i}\right].$$

Since $k(n-k)-(n-1)=n(k-1)-(k^2-1)=(k-1)(n-k-1)$, the previous expression is equivalent to

$$(n-1)(c_{pq})\left[\frac{(n-k-1)(k-1)}{n-1}\right]+(n-1)\left[\sum_{i=1}^{k-1} c_{qv_i}-c_{pv_i}\right].$$

Therefore, the difference between the communication cost of T^* and T is equal to

$$(n-1)\left[\sum_{i=1}^{k-1}\left(c_{qv_i}-c_{pv_i}+\frac{n-k-1}{n-1}c_{pq}\right)\right]. \tag{3.12}$$

Now, we obtain a lower bound on the value of (3.12). Since n is a constant and $c_e \geq 0$, we need to minimize $n-k-1$ as a function of k. Thus, its minimum value is attained when k is maximum. By the way we chose q, we have that $k \leq n/2$. So,

$$c_{qv_i}-c_{pv_i}+\frac{n-k-1}{n-1}c_{pq} \geq c_{qv_i}-c_{pv_i}+\frac{n-2}{2n-2}c_{pq} \geq 0.$$

The last inequality comes from the fact that $t \leq (n-2)/(2n-2)$ and $c_{ij}+tc_{jk} \geq c_{ik}$, for every $i,j,k \in V$. Therefore, T is also an optimal solution for $\mathcal{I}$, a contradiction.

□

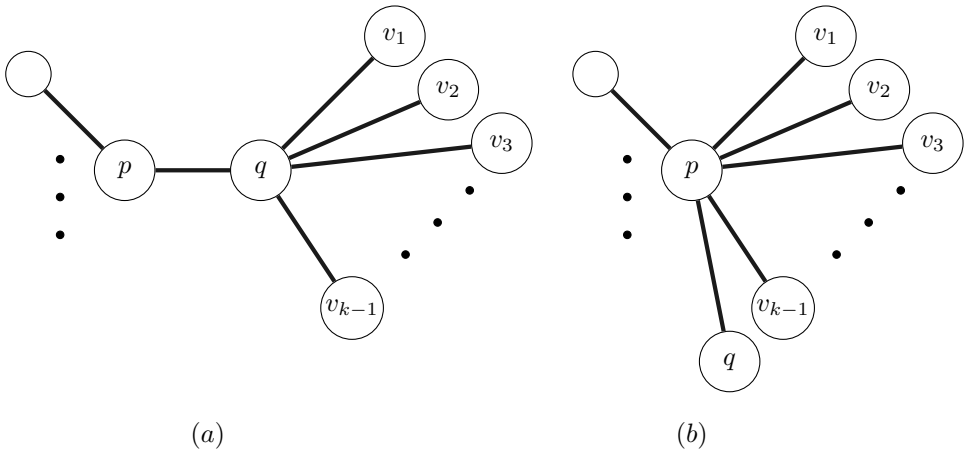


Figure 3.9: (a) *The tree* T^**;* (b) *the tree* T *stated in the proof of Theorem 3.4.10.*

Therefore, by Theorem 3.4.10, if an instance satisfies the conditions given in that theorem, we can find an optimal solution by considering each vertex as the center of the star, and return the one with minimum communication cost.

Chapter 4

Mixed Integer Linear Programming formulations

In this chapter, we first review some ILP and MILP formulations for the spanning tree problem. After that, we show how to separate an important class of inequalities, called the Subtour Elimination Contraints, that will be used in our implementation. Finally, we present the MILP formulations that exist in the literature for the OCST problem.

4.1 Formulations for the spanning tree problem

In the OCST problem, our aim is to find a spanning tree that minimizes the total communication cost. Thus, we first present formulations to find a spanning tree in a graph. Throughout this section we consider that the input graph is always connected (even if this is not stated explicitly).

There exist some well-known formulations for the spanning tree problem in the literature (see [MW95, CCPS98]). We will present three of these formulations. The first one is known as the Cut Set (CS) formulation, and the second one is known as the Subtour Elimination Constraint (SEC formulation). The third one models the problem as a flow in a network. These three models are classic and are mentioned in [MW95].

Let $G = (V, E)$ be the input graph for the spanning tree problem. In what follows, we consider binary variables $x = (x_e)_{e \in E}$ and impose restrictions to obtain a feasible solution (spanning tree) given by precisely the edges $e \in E$ such that $x_e = 1$.

4.1.1 Cut Set formulation

This formulation is based on the Cut Set inequalities (4.2). These inequalities impose that a feasible solution x must satisfy the following condition: $x(\delta(S)) \geq 1$, for each nonempty proper set $S \subset V$. That is, if T is the graph induced by the support of x, then T must contain an edge of every cut $\delta(S)$, defined by a nonempty proper subset $S \subset V$.

Since a spanning tree of G is a connected subgraph with $|V|-1$ edges, the spanning tree problem can be modelled by the system of inequalities (4.1), (4.2) and (4.3), indicated in what follows. We call this system the CS *formulation*.

$$x(E) = |V| - 1 \tag{4.1}$$

$$x(\delta(S)) \geq 1 \qquad \forall S \subsetneq V, \quad S \neq \emptyset \tag{4.2}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{4.3}$$

The next proposition shows that the above formulation models properly the spanning tree problem.

Proposition 4.1.1. *Let $G = (V, E)$ be a graph. A vector $x \in \mathbb{B}^{|E|}$ is a feasible solution of the* CS *formulation if, and only if, its support induces a spanning tree of G.*

Proof. First, observe that the incidence vector of any spanning tree satisfies (4.1), (4.2) and (4.3). We will show now that the support of a feasible solution x induces a spanning tree of G. Let T be the subgraph of G induced by the support of x. Since (4.1) implies that $|E(T)| = |V| - 1$, we just need to show that T is connected.

By contradiction, suppose that T is not connected. Then, T contains at least two connected components. Let S be the vertex set of one of these components. Clearly, S is nonempty, and $S \neq V$. Since there is no edge between S and $V \setminus S$, we have that $x(\delta(S)) = 0$, a contradiction to (4.2). □

Note that the CS formulation has an exponential number of inequalities. If we consider a relaxation of this formulation, it is possible to solve it in polynomial time, because the separation problem corresponding to (4.2) can be solved in polynomial time, by finding a minimum cut in the graph induced by the support of x. However, the relaxation of this formulation does not give an integral polytope (that is, we cannot drop the integrality restriction).

4.1.2 Subtour Elimination Constraint formulation

A spanning tree of $G = (V, E)$ is an acyclic subgraph containing $|V| - 1$ edges. The *Subtour Elimination Constraints* (see (4.5)) impose that the support of a feasible solution induce an acyclic subgraph. We call the system of inequalities (4.4), (4.5) and (4.6) the SEC *formulation*.

$$x(E) = |V| - 1 \tag{4.4}$$
$$x(E(S)) \leq |S| - 1 \quad \forall S \subsetneq V, S \neq \emptyset \tag{4.5}$$
$$x_e \in \{0, 1\} \quad \forall e \in E(G). \tag{4.6}$$

Proposition 4.1.2. *Let $G = (V, E)$ be a graph. A vector $x \in \mathbb{B}^{|E|}$ is a feasible solution of the* SEC *formulation if, and only if, its support induces a spanning tree of G.*

Proof. Let T be a spanning tree of G, and let χ^T be its incidence vector. Since any induced subgraph of T is a forest, then χ^T satisfies (4.5). Therefore, χ^T is a feasible solution of the SEC formulation. Let us now prove that, if x is a feasible solution, then the subgraph induced by the support of x is a tree. By Proposition 4.1.1, it suffices to show that x satisfies (4.2). Let S be a nonempty vertex set such that $S \subsetneq V$. Observe that

$$x(E) = x(E(S)) + x(E(V \setminus S)) + x(\delta(S)). \tag{4.7}$$

Since x satisfies (4.5), we have that $x(E(S)) \leq |S| - 1$ and $x(E(V \setminus S)) \leq |V| - |S| - 1$. By replacing these two inequalities in (4.7), we obtain that $x(E) \leq |V| - 2 + x(\delta(S))$. Finally, using equation (4.4) in the latter inequality, we have that $x(\delta(S)) \geq 1$. □

Similarly to the CS formulation, there exists an exponential number of inequalities (4.5). However, in this case, if we relax the integrality of this formulation, the corresponding polytope is integral. This means that if we can solve the separation problem corresponding to the inequalities (4.5), then we can solve the spanning tree problem in polynomial time (using this relaxed formulation). Fortunately, this separation problem can be solved in polynomial time. We prove this claim after we present the last formulation for the spanning tree problem.

4.1.3 Flow formulation

Let $G = (V, E)$ be the input graph, and let $D = (V, A)$ be its associated digraph. In the *Flow formulation*, we choose a vertex $r \in V$. Besides the variables $x = (x_e)_{e \in E}$, we also have variables $f = (f_a)_{a \in A}$ that represent a flow, in D, from r to every other vertex in V. Moreover, each arc a in this network has capacity $u_a = |V| - 1$, and we ensure that if $f_{ij} > 0$, then $x_{ij} = 1$.

The *Flow formulation* is defined by the following system of inequalities:

$$\sum_{ij \in E} x_{ij} = |V| - 1 \tag{4.8}$$

$$\sum_{ij \in A} f_{ij} - \sum_{jk \in A} f_{jk} = 1 \qquad \forall j \in V \setminus \{r\} \tag{4.9}$$

$$f_{ij} \leq (|V| - 1)x_{ij} \qquad \forall ij \in A \tag{4.10}$$

$$f_{ij} \geq 0 \qquad \forall ij \in A \tag{4.11}$$

$$x_{ij} \in \{0, 1\} \qquad \forall ij \in E \tag{4.12}$$

In what follows, we show that the Flow formulation models properly the spanning tree problem.

Proposition 4.1.3. *Let $G = (V, E)$ be a graph and let $D = (V, A)$ be its associated digraph. Let $x \in \mathbb{B}^{|E|}$, and let T be the subgraph of G induced by the support of x. Let $r \in V$. Then, T is a tree if and only if there exists a vector $f \in \mathbb{R}^{|A|}$ such that (x, f) is a feasible solution of the Flow formulation.*

Proof. First, we suppose that T is a tree, and show that there exists a vector f such that (x, f) is a feasible solution of the flow formulation. Let T^r be the r-arborescence obtained from T, and let $\mathcal{P}$ be the collection of paths, in T^r, from r to the other vertices in V. We denote by P_{ru} the path in $\mathcal{P}$ from r to u. We define f as follows. For each arc $a \in A$, we set

$$f_a = |\{P_{ru} : P_{ru} \in \mathcal{P},\, a \in A(P_{ru})\}|. \tag{4.13}$$

Since T is a tree, x satisfies (4.8). Note that, for each vertex $j \in V \setminus \{r\}$, if $a = ij$ is the arc in T^r entering j, then $f(a)$ is precisely the number of vertices in the j-arborescence, say T^j, contained in T^r, and this value is one unit greater than the number of vertices in $T^j - j$ (considering the definition of the flow value in each of the arcs leaving j). Thus, for each $j \in V \setminus \{r\}$, the pair(x, f) satisfies (4.9). Finally, (x, f) also satisfies (4.10) since $|\mathcal{P}| \leq |V| - 1$. Thus (x, f) is a feasible solution of the Flow formulation.

Now let (x, f) be a feasible solution of the Flow formulation. We will show that T is a tree. We denote by $f^+_-(u)$ the excess of flow in vertex u. Furthermore, if $S \subseteq V$, we denote by $f^+_-(S)$ the sum of $f^+_-(u)$ for each $u \in S$. By (4.9), we have that $f^+_-(u) = 1$, for each $u \in V \setminus \{r\}$. Moreover, by the flow conservation property, $f^+_-(r) = -|V| + 1$. By Proposition 4.1.1, it suffices to show that x satisfies the inequalities (4.2) in the CS formulation. Let S be a subset of V such that $S \neq \emptyset$ and $S \neq V$. Since $f^+_-(\tilde{S}) = 0$ only if $\tilde{S} = V$ or $\tilde{S} = \emptyset$, we have that $f^+_-(S) \neq 0$. Thus, there exists an arc a, from S to $V \setminus S$ or vice versa, such that $f_a > 0$. Then, by (4.10), we have that $x(\delta(S)) \geq 1$. Therefore, x satisfies (4.2). □

In Figure 4.1, we show an instance of the spanning tree problem. Besides that, we show the digraph induced by the support of a feasible solution (x, f) of the flow formulation.

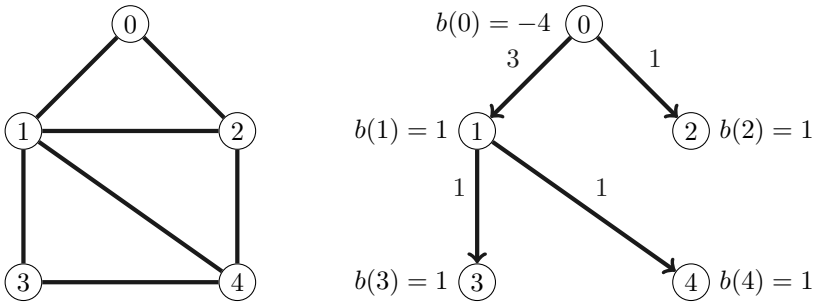


Figure 4.1: (*a*) *A graph G; and* (*b*) *an r-arborescence induced by the support of (x, f). The value $b(j)$ is the excess of flow in the node j.*

4.1.4 Separation routine for the SEC formulation

Let $G = (V, E)$ be the input graph, and consider the linear relaxation of the SEC formulation:

$$x(E) = |V| - 1, \tag{4.4}$$

$$x(E(S)) \leq |S| - 1 \qquad \forall S \subseteq V, S \neq \emptyset, \tag{4.5}$$

$$x_e \geq 0 \qquad \forall e \in E \tag{4.6}$$

Edmonds [Edm71] showed that the vertices of the polytope defined by these inequalities are integral. To solve the above relaxation in polynomial time, we need a polynomial algorithm –called here separation routine– to separate inequalities (4.5)

In what follows, given a vector x that satisfies (4.4) and (4.6), we show how to decide whether x satisfies (4.5) in polynomial time. If the negative case, a violated inequality is also shown. This algorithm is mentioned in [LRS11].

First, suppose that x does not satisfy (4.5). Moreover, let S' be a subset $S' \subseteq V$, $S' \neq \emptyset$ that maximizes

$$x(E(S')) - |S'| + 1 > 0. \tag{4.14}$$

Using the equation $1 = |V| - x(E)$ in the previous inequality, we obtain

$$x(E) - x(E(S')) + |S'| < |V|.$$

Observe that $x(E) - x(E(S')) = x(E(V \setminus S')) + x(\delta(S'))$. Thus, if S' maximizes the left-hand side of (4.14), it minimizes

$$x(E(V \setminus S')) + x(\delta(S')) + |S'|. \tag{4.15}$$

Therefore, our problem reduces to finding a set $S' \subseteq V$, $S' \neq \emptyset$, that minimizes (4.15) and check if this value is less than $|V|$. In what follows, we model this problem as a minimum cut problem in a particular network.

Let $D = (V, A)$ be the associated digraph of G. Let s and t be different vertices of V, and consider the following sets of arcs:

$$\begin{aligned} A_s &= \{(s,p) : p \in V \setminus \{s\}\}, \\ A_t &= \{(p,t) : p \in V \setminus \{t\}\}. \end{aligned}$$

Let $D_{st} = (V, A \cup A_s \cup A_t)$. Let (D_{st}, u) be a network, where the capacity u is defined as follows.

$$u_{ij} := \begin{cases} \frac{x_{ij}}{2}, & \text{if } ij \in A \\ \frac{x(\delta(j))}{2}, & \text{if } ij \in A_s \\ 1, & \text{if } ij \in A_t. \end{cases} \tag{4.16}$$

Note that, there can be more than one arc between each pair of vertices. Let $S \subseteq V$ such that $s \in S$ and $t \notin S$. We denote by $\delta^+_{st}(S)$ the cut induced by S in D_{st}. In order to calculate the value of $u(\delta^+_{st}(S))$, we consider the following three sets:

$$\begin{aligned} A_1 &= \{ij \in A : i \in S,\, j \notin S\}, \\ A_2 &= \{it \in A_t : i \in S\}, \\ A_3 &= \{si \in A_s : i \notin S\}. \end{aligned}$$

Note that $u(\delta^+_{st}(S)) = u(A_1) + u(A_2) + u(A_3)$. Moreover, we have that $u(A_1) = x(\delta(S))/2$ and $u(A_2) = |S|$. Finally,

$$\begin{aligned} u(A_3) &= \sum_{i \in V \setminus S} x(\delta(i))/2 \\ &= \tfrac{1}{2}\left(2x(E(V \setminus S)) + x(\delta(V \setminus S))\right) \\ &= x(E(V \setminus S)) + x(\delta(V \setminus S))/2 \\ &= x(E(V \setminus S)) + x(\delta(S))/2. \end{aligned}$$

Therefore, we obtain that

$$u(\delta^+_{st}(S)) = x(E(V \setminus S)) + x(\delta(S)) + |S|.$$

The following observation will reduce the number of minimum cut computations needed to find a set S' that minimizes (4.15). Let $S \subseteq V$, and let $s, s' \in S$ and $t, t' \notin S$. Let (D_{st}, u)

(resp. $(D_{s't'}, u')$) be the network constructed from s and t (resp. s' and t') as above. Note that, for $i \in V \setminus S$, the arc $si \in A_s$ has the same capacity as the arc $s'i \in A_{s'}$. In a similar way, for $i \in S$, the arc $it \in A_t$ has the same capacity as the arc $it' \in A_{t'}$. This implies that $u(\delta^+_{st}(S)) = u'(\delta^+_{s't'}(S))$. Thus, to find a set $S' \subseteq V$, $S' \neq \emptyset$, we do the following:

1. choose a vertex $s \in V$,
2. for each $t \in V \setminus \{s\}$, construct D_{st} and find a minimum s-t cut,
3. for each $t \in V \setminus \{s\}$, construct D_{ts} and find a minimum t-s cut.

Therefore, to check if a vector x satisfies (4.5), we need to solve $2|V| - 2$ minimum cut computations.

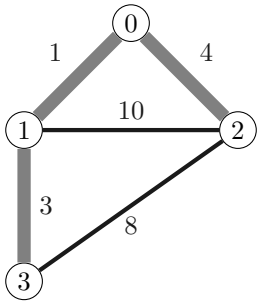
4.2 Mixed ILP formulations for the OCST problem

4.2.1 Path-Based formulation

Contreras, Fernández and Marín [CFM10a] proposed the Path-Based (PB) formulation. In what follows, we will describe the idea behind this formulation.

Let $\mathcal{I} = (G, c, R, w)$ be an instance of the OCST problem, where $G = (V, E)$. For each requirement $r \in R$, we consider the digraph $D = (V, A)$ (associated with G and r), in which we want to find a path that links (r_o, r_d).

In Figure 4.2 we show an instance of the OCST problem. Moreover, in the same figure, we indicate with ticker edges a solution, say T, for such instance. Besides that, in Figure 4.3 we show the communication paths, for T, between each origin and destination pair.



Requirement (r)	**Demand** (w_r)
$(0, 1)$	4
$(2, 1)$	2
$(2, 3)$	1

Figure 4.2: *An instance of the OCST problem and a feasible solution T, indicated with ticker edges.*

We consider a binary variable $x = (x_e)_{e \in E}$ such that $x_e = 1$ if, and only if, e belongs to our solution, say T. Moreover, we consider a binary variable $y = (y^r_a)$, where $r \in R$ and $a \in A(D)$, such that $y^r_a = 1$ if, and only if, a belongs to the path $P_{(r_o, r_d)}$ in T. Note that we are not considering SEC because we modify the set R, as described in the next paragraphs, to ensure connectivity.

The *Path-Based (PB) formulation* is defined by the following mixed integer linear program:

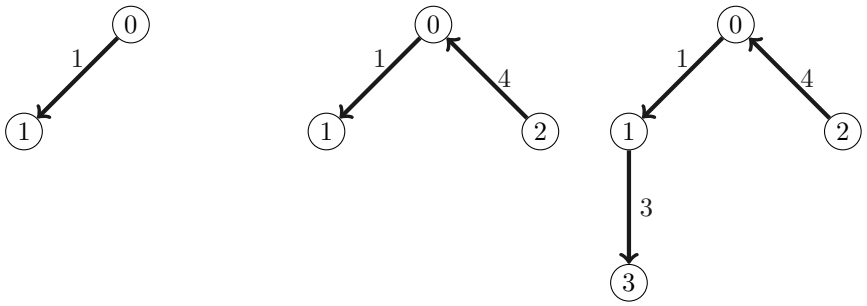


Figure 4.3: *All communication paths for R, modelled as network flows, in T.*

$$\min \quad \sum_{r \in R} w_r \sum_{ij \in A} c_{ij} y^r_{ij} \tag{4.17}$$

s.t.

$$\sum_{ij \in E} x_{ij} = |V| - 1 \tag{4.18}$$

$$\sum_{i, r_d \in A} y^r_{ir_d} = 1 \qquad \forall r \in R \tag{4.19}$$

$$\sum_{r_o, k \in A} y^r_{r_o k} = 1 \qquad \forall r \in R \tag{4.20}$$

$$\sum_{ij \in A} y^r_{ij} - \sum_{jk \in A} y^r_{jk} = 0 \qquad \forall r \in R \quad \forall j \in V \setminus \{r_o, r_d\} \tag{4.21}$$

$$y^r_{ij} + y^r_{ji} \leq x_{ij} \qquad \forall r \in R \quad \forall ij \in E \tag{4.22}$$

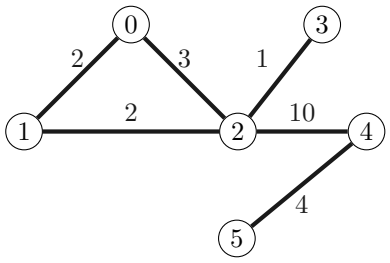
$$x_{ij} \in \{0, 1\} \quad \forall ij \in E \tag{4.23}$$

$$y^r_{ij} \geq 0 \qquad \forall r \in R \quad \forall ij \in A \tag{4.24}$$

Constraints (4.19), (4.20), and (4.21) model the path between the origin and destination of each requirement $r \in R$ as a unit flow. Moreover, constraint (4.22) ensures that the support of y^r induces a path in D that links r_o and r_d.

Figure 4.4 shows an instance (G, c, R, w) of the OCST problem, where $G = (V, E)$ is the graph depicted and the requirements $r \in R$ are indicated in the table. Besides that, in Figure 4.5, we show a solution that is not connected. Observe that, adding an artificial requirement $(0, 4)$ to R with $w_{04} = 0$ will guarantee the connectivity of our solution. So, we will fix a vertex o in V and for each vertex $v \neq o$ in V, we add the pair (o, v) to R with $w_{o,v} = 0$ if such a pair does not exist. This ensures that there exists a path between each pair of vertices. So, T is connected. Therefore, by constraint (4.18), T is a tree.

According to Luna [Lun16], the PB formulation produces the tightest linear relaxation bound. Despite that, because of the huge number of variables and constraints (up to $O(n^4)$), a good performance is only achieved when the instances are small or medium.



Requirement (r)	Demand (w_r)
$(0, 1)$	4
$(2, 1)$	2
$(2, 0)$	1
$(4, 5)$	3

Figure 4.4: *An instance of the OCST problem.*

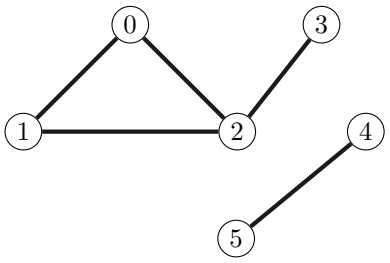
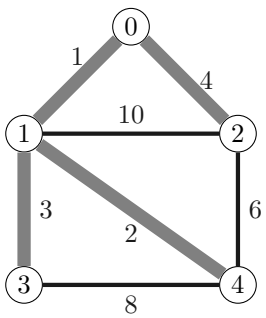


Figure 4.5: *A graph T that is not connected, induced by the support of an optimal solution for the PB formulation.*

4.2.2 Flow-Based formulation

In the Flow-Based (FB) formulation, the OCST problem is modelled as a set of flows in a tree. This idea was proposed by Fernández et al. [FLH+13]. Given an instance (G, c, R, w) of the OCST problem, where $G = (V, E)$, we want to find a subgraph H of G such that H is a tree. Moreover, for each $o \in V$, we obtain an o-arborescence contained in $D(H)$, the digraph associated with H. Throughout each one of these arborescences, we send flow from o to the other vertices.

Consider the instance (G, c, R, w) shown in Figure 4.6. To explain how to calculate the communication cost of a tree, we will consider the spanning tree T, composed of the ticker edges $\{0, 1\}$, $\{0, 2\}$, $\{1, 3\}$ and $\{1, 4\}$.



Requirement (r)	Demand (w_r)
$(0, 1)$	4
$(0, 3)$	3
$(2, 1)$	2
$(2, 4)$	1
$(3, 4)$	5

Figure 4.6: *An instance of the OCST problem and a feasible solution T, indicated with ticker edges.*

In this case, note that R contains three different origins for all requirements; these are the vertices 0, 2, and 3. To calculate the communication cost of a tree, each edge ij in the

tree contributes to the total communication cost with the value:

$$\sum_{ij \in P_{(r_o, r_d)}} c_{ij} w_r.$$

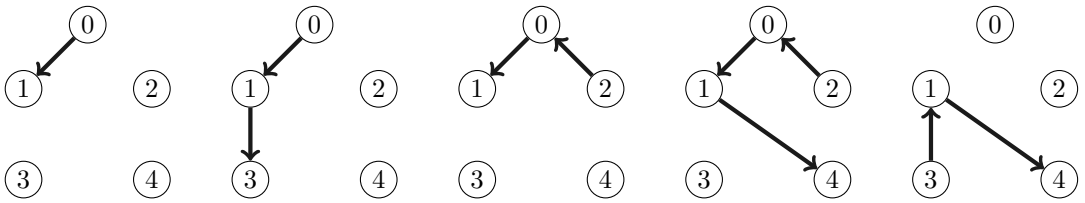


Figure 4.7: *All communication paths in T.*

We calculate now the communication cost of T, beginning with requirements that have their origin at vertex 0. As edge $\{0, 1\}$ is in the path from requirements with destinations 1 and 3, then its contribution is $c_{01}(w_{01} + w_{03}) = 1(4+3) = 7$; edge $\{1, 3\}$ is in the path to the requirement with destination 3, then its contribution is $c_{13}w_{03} = 3(3) = 9$. Thus, their total contribution is $7+9 = 16$ for all requirements with origin 0. The remaining edges contribute 0 as they do not take part in any communication path with origin 0. Similarly, we conclude that communication paths starting from 2 and 3 contribute 8 and 25, respectively. We can observe the network flows associated with this example in Figure 4.8.

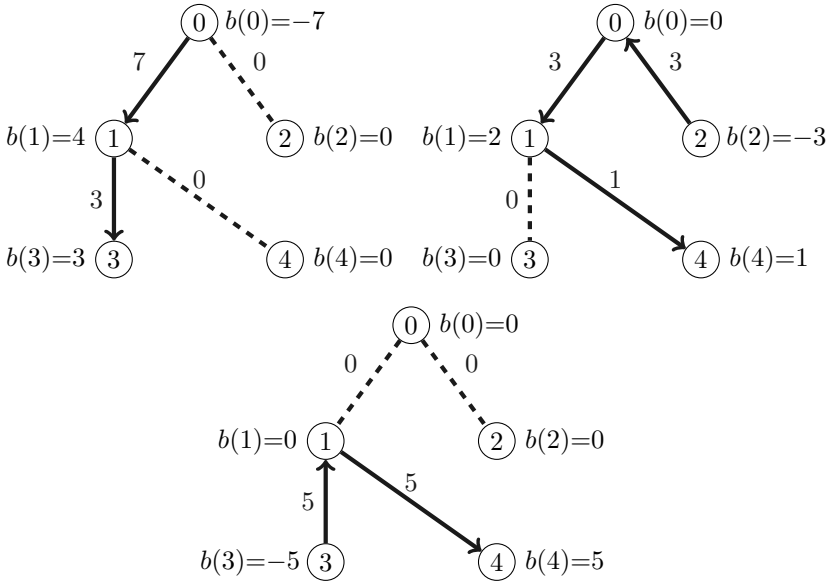


Figure 4.8: *All communication paths in T as network flows.*

More formally, we will consider a binary variable $x = (x_e)_{e \in E}$, such that $x_e = 1$ if, and only if, e belongs to H. Besides that, for each vertex $o \in V$, we have a variable $y = (y_a^o)$, such that $y_{ij}^o = 1$ if, and only if, ij belongs to the o-arborescence. Finally, variable f_{ij}^o indicates the amount of flow in the arc ij that belongs to the network associated with origin o.

The *Flow-Based (FB) formulation* is defined by the following mixed integer linear program:

$$\min \quad \sum_{o \in V} \sum_{ij \in A} c_{ij} f^o_{ij} \tag{4.25}$$

$s.t.$

$$\sum_{ij \in E} x_{ij} = |V| - 1 \tag{4.26}$$

$$\sum_{ij \in \mathbf{E}(S)} x_{ij} \leq |S| - 1 \qquad \forall S \subset V, \quad S \neq \emptyset \tag{4.27}$$

$$\sum_{ij \in A} f^o_{ij} - \sum_{jk \in A} f^o_{jk} = w_{oj} \qquad \forall o \in V \quad \forall j \in V \setminus \{o\} \tag{4.28}$$

$$\sum_{ok \in A} f^o_{ok} = \sum_{(o,d) \in R} w_{od} \qquad \forall o \in V \tag{4.29}$$

$$f^o_{ij} \leq M y^o_{ij} \qquad \forall o \in V \quad \forall ij \in A \tag{4.30}$$

$$\sum_{ij \in A} y^o_{ij} = |V| - 1 \qquad \forall o \in V \tag{4.31}$$

$$y^o_{ij} + y^o_{ji} \leq x_{ij} \qquad \forall o \in V \quad \forall ij \in E \tag{4.32}$$

$$f^o_{ij} \geq 0 \qquad \forall o \in V \quad \forall ij \in A \tag{4.33}$$

$$y^o_{ij} \in \{0, 1\} \qquad \forall o \in V \quad \forall ij \in A \tag{4.34}$$

$$x_{ij} \in \{0, 1\} \qquad \forall ij \in E \tag{4.35}$$

Note that the FB formulation can be seen as $|V|$ instances of the 1-source OCST problem. Constraints (4.26) and (4.27) ensure that the support of x induces a spanning tree, say H. Besides that, constraints (4.31) and (4.32) guarantee that the digraph induced by the support of y^o is an o-arborescence contained in $D(H)$.

The family of constraints (4.29) guarantees that the initial flow going out from o is equal to the sum of all demands with its origin at o. Constraints (4.30) and (4.32) ensure that if variable x_{ij} is activated, only one of the arcs ij or ji may have a positive flow. Constraints (4.28) are the demands of the vertices. These constraints guarantee that the flow retained by a vertex j is equal to w_{oj}. Constraints (4.29) guarantee that each vertex chosen as origin is the unique source for its associated network (negative demand).

Note that, to guarantee connectedness, it is not enough to add artificial requirements to R with demand zero. Therefore, contrary to the PB formulation, we consider the Subtour Elimination Constraints (4.27) to guarantee the connectedness of H.

Relaxation of the Flow-Based formulation

In the Flow-Based formulation, the set of feasible solutions correspond to arborescences. If we relax the problem to find digraphs whose underlying graph corresponds to a tree, the

set of optimal solutions is the same, as we will show later. Note that this relaxation is on the set of feasible solutions. Thus, we will call this formulation the *Relaxation of the Flow Based (RFB) formulation.*

Given an instance $\mathcal{I} = (G, c, R, w)$ of the OCST problem, we want to find a spanning tree T of G. For that, for each vertex o we obtain a network, such that we distribute flow from o to the remaining vertices. Each variable f^o_{ij} indicates the quantity of flow in the arc a_{ij} that belongs to the network associated to origin o.

The *RFB formulation* was introduced by [Lun16] in 2016. It is precisely the following mixed integer linear program:

$$\min \quad \sum_{o \in V} \sum_{ij \in A} c_{ij} f^o_{ij} \tag{4.36}$$

$$s.t.$$

$$\sum_{ij \in E} x_{ij} = |V| - 1 \tag{4.37}$$

$$\sum_{ij \in \mathbf{E}(S)} x_{ij} \leq |S| - 1 \qquad \forall S \subset V, \quad S \neq \emptyset \tag{4.38}$$

$$\sum_{ij \in A} f^o_{ij} - \sum_{jk \in A} f^o_{jk} = w_{oj} \qquad \forall o \in V \quad \forall j \in V \setminus \{o\} \tag{4.39}$$

$$\sum_{ok \in A} f^o_{ok} = \sum_{(o,d) \in R} w_{od} \qquad \forall o \in V \tag{4.40}$$

$$f^o_{ij} + f^o_{ji} \leq \left(\sum_{(o,d) \in R} w_{od} \right) x_{ij} \qquad \forall o \in V \quad \forall ij \in E \tag{4.41}$$

$$f^o_{ij} \geq 0 \qquad \forall o \in V \quad \forall ij \in A \tag{4.42}$$

$$x_{ij} \in \{0, 1\} \qquad \forall ij \in E \tag{4.43}$$

Constraints (4.37) and (4.38) ensure that the support of x induces a spanning tree. Constraints (4.40) guarantee that the initial flow going out from o is equal to the sum of the demands of all requirements that have origin at o. Constraints (4.41) ensure that if any of the arcs ij or ji has a nonzero flow, the associated variable x_{ij} must be activated. Nevertheless, both arcs can have a nonzero flow. Constraints (4.39) guarantee that the flow retained by a vertex j in the network associated with o is equal to w_{oj}.

In contrast with the FB formulation, we do not consider variables y. However, if an optimal solution (x, f) for this MILP exists, each digraph induced by f^o variables is an o-arborescence, as shown in Proposition 4.2.1.

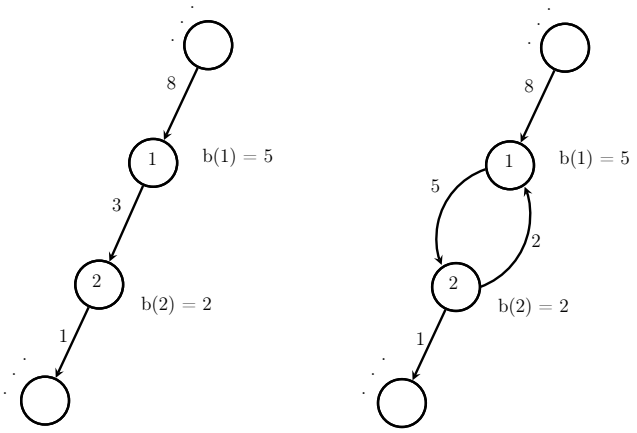


Figure 4.9: *Procedure to delete arcs that form a cycle and still get a feasible solution.*

Proposition 4.2.1. *Let $\mathcal{I} = (G, c, R, w)$ be an instance of the OCST problem and let o be any vertex of G. Let (x, f) be an optimal solution of the RFB formulation for the instance $\mathcal{I}$, if the problem is feasible. Let T be the graph induced by the support of x and let T' be the digraph induced by the support of f^o. Then the digraph T' is an o-arborescence contained in $D(G)$.*

Proof. First, observe that T is the underlying graph of T'. Now suppose, by contradiction, that T' contains opposite arcs. Consider two opposite arcs in T', say a and d ($f_a \leq f_d$), we can construct a new feasible solution (x, f'), where f' is defined, for each arch $a' \in T$, in the following way

$$f'_{a'} := \begin{cases} 0, & \text{if } a' = a \\ f_d - f_a, & \text{if } a' = d \\ f_{a'}, & \text{otherwise.} \end{cases} \tag{4.44}$$

Clearly, (x, f') is a feasible solution with a smaller cost than the cost of (x, f), contradicting that (x, f) is optimal.

Since T is a tree and each vertex u of T' different than o satisfies $f^+_-(u) \leq 0$, it follows that T' is an o-arborescence. □

The following result follows as a corollary of the previous statement.

Corollary 4.2.2. *Each digraph induced by f^o is an o-arborescence. Moreover, the optimal value of the RFB formulation is equal to the communication cost of an OCST.*

Since we consider n origins and the number of edges can be at most n^2, the FB formulation and its relaxation have at most $O(n^3)$ variables.

4.2.3 Rooted Tree based formulation

In the *Rooted Tree Based (RTB) formulation*, proposed by [Lun16] to solve the OCST problem, the idea is to find an o-arborescence, for a fixed vertex o (called root).

Let (G, c, R, w) be an instance of the OCST problem, and $D = (V, A)$ its associated digraph. We want to obtain a directed subgraph T of D such that T is an o-arborescence for a fixed vertex $o \in V$. Besides that, we want to find the distance for each pair of vertices of G in the underlying graph of T.

We show first how to model this problem (of finding an o-arborescence), which can be seen as a model to find a spanning tree. We consider two sets of variables. We use binary variables $x = (x_a)_{a \in A}$ to indicate which arcs of D are selected to belong to our solution. More precisely, $x_a = 1$ if and only if $a \in A$ is selected. Besides that, we consider binary variables $p = (p_{ij})_{ij \in V^2}$ to indicate whether there is a directed path from vertex i to vertex j in our solution. (This is needed to deal later with the distance between each pair ij of variables.)

The *Rooted Tree formulation* is defined by the following system of inequalities.

$$\sum_{io \in A} x_{io} = 0 \tag{4.45}$$

$$\sum_{ij \in A} x_{ij} = 1 \qquad \forall j \in V \setminus \{o\} \tag{4.46}$$

$$x_{ij} \leq p_{ij} \qquad \forall ij \in A \tag{4.47}$$

$$p_{ij} + p_{ji} \leq 1 \qquad \forall ij \in V^2 \tag{4.48}$$

$$p_{ij} + x_{jk} \leq 1 + p_{ik} \qquad \forall \{i, j, k\} \in V^3, i \neq j, j \neq k \tag{4.49}$$

$$p_{ik} + x_{jk} \leq 1 + p_{ij} \qquad \forall \{i, j, k\} \in V^3, i \neq j, j \neq k \tag{4.50}$$

$$x_{ij} \in \{0, 1\} \qquad \forall ij \in A \tag{4.51}$$

$$p_{ij} \in \{0, 1\} \qquad \forall ij \in V^2 \tag{4.52}$$

Proposition 4.2.4 shows that if (x, p) is a feasible solution of the above formulation, then the support of x defines an o-arborescence T and the variable p satisfies: $p_{ij} = 1$ if, and only if, there exists a path from i to j in T.

Proposition 4.2.3. *Let $D = (V, A)$ be a digraph, and let $\chi \in \mathbb{B}^{|A|}$ be its incidence vector. Let p be a vector in $\mathbb{B}^{|V| \times |V|}$. Suppose that (χ, p) satisfies (4.47) and (4.49). Then $p_{ik} = 1$ if there is a path from i to k in D.*

Proof. Let P_{ik} be a path from i to k in D. We show that $p_{ik} = 1$ by induction on $|P_{ik}|$. If $|P_{ik}| = 1$, then $\chi_{ik} = 1$. Since (χ, p) satisfies (4.47), we have that $p_{ik} = 1$.

Now suppose that $|P_{ik}| \geq 2$. Consider that $P_{ik} = \langle i, \ldots, j, k \rangle$. By the induction hypothesis, we have that $p_{ij} = 1$. Moreover, since $\chi_{jk} = 1$, the inequality (4.49) implies that $p_{ik} = 1$. □

We show now that every feasible solution of the Rooted Tree formulation induces an o-arborescence.

Proposition 4.2.4. *Let $G = (V, E)$ be a graph and let $D = (V, A)$ be its associated digraph. Consider any fixed vertex o of G. Let $x \in \mathbb{B}^{|A|}$ and let T be the digraph induced by the support of x. The digraph T is an o-arborescence if, and only if, there exists a vector p in $\mathbb{B}^{|V|\times|V|}$ such that (x, p) is a feasible solution of the Rooted Tree formulation with root o. Moreover, for vertices i and j in T, we have that $p_{ij} = 1$ if, and only if, there exists a path from i to j in T.*

Proof. First, suppose that T is an o-arborescence. We define p as follows:

$$p_{ij} = \begin{cases} 1, & \text{if there exists a path in } T \text{ from } i \text{ to } j. \\ 0, & \text{otherwise.} \end{cases}$$

Since T is an o-arborescence, (x, p) satisfies (4.45) and (4.46). The definition of p implies inequality (4.47). Moreover, since T is acyclic, (x, p) satisfies (4.48). Let i, j and k be vertices of T. Observe that (4.49) is satisfied when $p_{ij}+x_{jk} \leq 1$. So, suppose that $p_{ij} = 1$ and $x_{jk} = 1$. From the definition of p, there exists a path from i to j; and $x_{jk} = 1$ implies that there is a path from j to k. Thus, there exists a path from i to k (so $p_{ik} = 1$). This implies that (x, p) satisfies (4.49). By an analogous argument, (x, p) also satisfies (4.50). Therefore, (x, p) is a feasible solution for the Rooted Tree formulation.

Now let (x, p) be a feasible solution of the Rooted Tree formulation. Let T be the digraph induced by the support of x. We will show that T is an o-arborescence. In what follows, we show that there is a path, in T, from o to every other vertex in V. Let $k \in V \setminus \{o\}$, and let P_k be a longest path in T ending at k. Let i be the vertex where P_k begins. By the way P_k was chosen, either $i = o$ or there is an arc ji such that $j \in V(P_k) \setminus \{k\}$. In the latter case, by Proposition 4.2.3, we have that $p_{ij} + p_{ji} = 2$, a contradiction to (4.48). This implies that $i = o$. Using the fact that T contains a path from o to every vertex in V, combined with inequalities (4.46)–(4.48) and Proposition 4.2.3, we conclude that T is an o-arborescence.

Finally, let i and k be vertices of T. If T has a path from i to k, then by Proposition 4.2.3, we have that $p_{ik} = 1$. Suppose now that there is no path in T from i to k. We shall prove that $p_{ik} = 0$. Let j^* be the lowest common ancestor of i and k. If $j^* = k$, then there is a path from k to i in T, and (as we have shown), $p_{ki} = 1$. Thus, from (4.48), we have that $p_{ik} = 0$. Let us now consider that $j^* \neq k$. Suppose $p_{ik} = 1$. Let j be the parent of vertex k in the o-arborescence T. Then $x_{jk} = 1$. From (4.50) we conclude that $p_{ij} = 1$. Let $j^* = j_p, j_{p-1}, \ldots, j_1 = k$ be the vertices in the path from j^* to k in the o-arborescence T. Repeating for each of the vertices j_i, for $i = 3, \ldots, p$ the same argument we have used for j $(= j_2)$, we conclude that $p_{ij^*} = 1$. Since there is a path from j^* to i in T, we have that $p_{j^*i} = 1$. But then, $p_{j^*i} + p_{ij^*} = 2$, a contradiction to (4.48). Thus, $p_{ik} = 0$. This concludes the proof of the proposition. □

The *Rooted Tree Based (RTB) formulation* proposed for the OCST problem is obtained from the previous (Rooted Tree) formulation extended with variables d_{ij} to capture the distance between i and j in the tree (defined by the support of x).

The *RTB formulation* is the following mixed integer linear program:

$$\min \sum_{r \in R} w_r d_r \tag{4.53}$$

$$s.t.$$

$$\sum_{ij \in A} x_{ij} = 1 \qquad \forall j \in V \setminus \{o\} \tag{4.54}$$

$$x_{ij} \leq p_{ij} \qquad \forall ij \in A \tag{4.55}$$

$$p_{ij} + p_{ji} \leq 1 \qquad \forall ij \in V^2 \tag{4.56}$$

$$p_{ij} + x_{jk} \leq 1 + p_{ik} \qquad \forall \{i,j,k\} \in V^3, i \neq j, j \neq k \tag{4.57}$$

$$p_{ik} + x_{jk} \leq 1 + p_{ij} \qquad \forall \{i,j,k\} \in V^3, i \neq j, j \neq k \tag{4.58}$$

$$d_{ik} + c_{kj} - M(2 - x_{kj} - p_{ik}) \leq d_{ij} \qquad \forall \{i,j,k\} \in V^3, i \neq j, j \neq k \tag{4.59}$$

$$d_{ik} + c_{kj} - M(1 - x_{kj} + p_{ij} + p_{ji}) \leq d_{ij} \qquad \forall \{i,j,k\} \in V^3, i \neq j, j \neq k \tag{4.60}$$

$$d_{ij} \geq c_{ij}(x_{ij} + x_{ji}) \qquad \forall ij \in A \tag{4.61}$$

$$x_{ij} \in \{0,1\} \qquad \forall ij \in A \tag{4.62}$$

$$p_{ij} \in \{0,1\} \qquad \forall ij \in V^2 \tag{4.63}$$

To calculate the optimal communication cost in this formulation, we consider the distances between each origin/destination requirement $r \in R$ multiplied by its demand w_r. Therefore, we obtain the objective function (4.53).

The constant M is an upper bound for the cost of each path of G. Since finding a longest (cost) path is an NP-hard problem [KMR97], in our implementation we consider M as the maximum cost of an edge of G times $|V| - 1$.

Proposition 4.2.5 shows that the RTB formulation is a valid formulation for the OCST problem.

Proposition 4.2.5. *Let $\mathcal{I} = (G, c, R, w)$ be an instance of the OCST problem and let o be a vertex of G. Let x be a vector in $\mathbb{B}^{|A|}$. Let T be the digraph induced by the support of x. The digraph T is an o-arborescence if, and only if, there exist two vectors $p \in \mathbb{B}^{|V| \times |V|}$ and $d \in \mathbb{R}^{|V| \times |V|}$ such that (x, p, d) is a feasible solution of the* RTB *formulation with root o. Besides that, if (x, p, d) is an optimal solution, its objective value is equal to the communication cost of the underlying graph of T.*

Proof. By Lemma 4.2.4, such p exists if, and only if, T is an o-arborescence. Thus, we prove the statement by showing that, when p exists, for each pair of vertices u and v, d_{uv} is lower bounded by the minimum distance between u and v in T. First, note that for any pair of vertices u and v, $d_{uv} \geq c_{uv}$ when $x_{uv} = 1$. Now let i, j and k be vertices of G. Note that constraint (4.59) is trivially satisfied when $x_{kj} + p_{ik} < 2$. Clearly, constraint (4.60) is trivially satisfied when $p_{ij} + p_{ji} = 1$ or $x_{kj} = 0$. Thus, we suppose that $x_{kj} = 1$ or, equivalently, that k is the parent of j in T. Without loss of generality, we now distinguish two cases. In the first one, i is an ancestor of j in T (see Figure 4.10). Since $x_{kj} = 1$ an $p_{ik} = 1$, we have that $d_{ik} + c_{kj} \leq d_{ij}$. In the second case, i and j have a common ancestor different from i and j (see Figure 4.11). Since T is acyclic we have that $p_{ij} + p_{ji} = 0$. Therefore, $d_{ik} + c_{kj} \leq d_{ij}$.

□

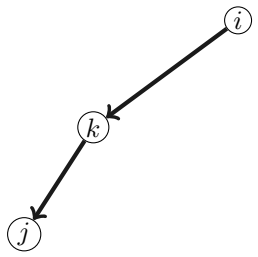


Figure 4.10: $d_{ik} + c_{kj} - M(2 - x_{kj} - p_{ik}) \leq d_{ij}$.

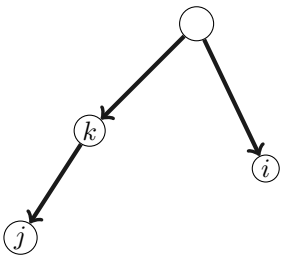


Figure 4.11: $d_{ik} + c_{kj} - M(1 - x_{kj} + p_{ij} + p_{ji}) \leq d_{ij}$.

4.3 Other MILP formulations for the OCST problem

We conclude this chapter by mentioning the current exact algorithm for OCST with better performance. Before, we mention two other approaches based on Dantzig-Wolfe decomposition.

Solving instances with more than 40 vertices using Path and Flow Based formulations takes too much time (owing to their large number of variables and constraints). To overcome the weakness of these formulations, Tilk and Irnich [TI18] presented a Dantzig–Wolfe Decomposition for such formulations. Both Dantzig–Wolfe reformulations are solved within a branch-and-price-and-cut framework.

More recently, Zetina, Contreras, Fernández and Luna [ZCFL19] proposed an algorithm that uses a strong Benders reformulation within a branch-and-cut framework. To our knowledge, this is currently the best exact algorithm for OCST. It manages to solve instances with up to 60 vertices.

4.3.1 Dantzig-Wolfe decomposition for the Path-Based formulation

The Path-Based formulation can be decomposed by each requirement $r \in R$ so that all y_{ij}^r variables for a fixed r form a block. Using this decomposition, the Dantzig–Wolfe reformulation replaces the y_{ij}^r variables by variables representing communication paths, keeps the x variables and the constraints (4.20) and (4.21) (constraints (4.19) are ignored by flow conservation properties) in the master problem, and reformulates constraints (4.22) with path variables.

Let $\mathcal{P}_r$ be the set of directed paths from r_o to r_d for each $r \in R$, and let $\overline{c}_r^p$ be the communication cost of a path p for a requirement r. We consider the variables λ_r^p for each $r \in R$ and $p \in \mathcal{P}_r$. Besides that, we consider variables $\overline{y}_{r,ij}^p$ is equal to one if path $p \in \mathcal{P}_r$ contains the arc ij.

The integer master problem of the Dantzig–Wolfe reformulation of the PB formulation is the following:

$$\min \quad \sum_{r\in R}\sum_{p\in\mathcal{P}_r} \overline{c}_r^{\,p}\lambda_r^p \tag{4.64}$$

s.t.

$$\sum_{p\in\mathcal{P}_r} \left(\overline{y}^p_{(r,ij)} + \overline{y}^p_{(r,ji)}\right)\lambda_r^p \le x_{ij} \qquad \forall r\in R \quad \forall ij\in E \tag{4.65}$$

$$\sum_{ij\in E} x_{ij} = |V| - 1 \tag{4.66}$$

$$\sum_{p\in\mathcal{P}_r} \lambda_r^p = 1 \qquad \forall r\in R \tag{4.67}$$

$$x_{ij}\in\{0,1\} \qquad \forall ij\in E \tag{4.68}$$

$$\lambda_r^p \ge 0 \qquad \forall r\in R \quad \forall p\in\mathcal{P}_r \tag{4.69}$$

Constraints (4.67) are the convexity constraints forcing the selection of exactly one path from r_o to r_d for each $r\in R$. Let $\pi_{ij}^r \le 0$ be the dual prices of the constraints (4.65) of the restricted master problem, and let $\mu_{st}\in\mathbb{R}$ be the dual prices of the convexity constraints (4.67). The pricing problem (to look for a negative reduced cost), for each block associated with a fixed r, is the following:

$$\min \quad \sum_{ij\in A} w_r c_{ij} - \pi_{ij}^r \overline{y}_{ij}^r - \mu_r \tag{4.70}$$

s.t.

$$\sum_{r_o,j\in A} \overline{y}_{sj}^r = 1 \tag{4.71}$$

$$\sum_{ij\in A} \overline{y}_{ij}^r - \sum_{ji\in A} \overline{y}_{ji}^r = 0 \qquad \forall i\in V\setminus\{r_o,r_d\} \tag{4.72}$$

$$\overline{y}_{ij}^r \ge 0 \qquad \forall ij\in A \tag{4.73}$$

Tilk and Irnich [TI18] solve the pricing problem using Dijkstra's algorithm [Dij59].

4.3.2 Dantzig-Wolfe decomposition for the Flow-Based formulation

The Flow-Based formulation can be decomposed by each vertex $u\in V$ so that all y_{ij}^u and f_{ij}^u variables for a fixed u form one block.

Let $\mathcal{S}_u$ be the set of all (spanning) u-arborescences for each $u\in V$. We consider the variables λ_u^q for each $u\in V$ and $q\in\mathcal{S}_u$. We also consider variables $\overline{f}^{\,q}_{u,ij}$ that is the flow in

arc ij for the u-arborescence $q \in \mathcal{S}_u$ and variables $\overline{y}^q_{u,ij}$ is equal to one if the tree $q \in \mathcal{S}_u$ contains the arc ij. Let $\overline{c}^q_u$ be the communication cost of the u-arborescence $q \in \mathcal{S}_u$.

The master problem of the Dantzig-Wolfe decomposition of the Path-Based formulation is the following:

$$\min \quad \sum_{u \in V} \sum_{p \in \mathcal{Q}_u} \overline{c}^q_u \lambda^q_u \tag{4.74}$$

s.t.

$$\sum_{q \in \mathcal{Q}_u} \left(\overline{y}^q_{(u,ij)} + \overline{y}^q_{(u,ji)}\right) \lambda^q_u \leq x_{ij} \qquad \forall u \in V \quad \forall ij \in E \tag{4.75}$$

$$\sum_{ij \in E} x_{ij} = |V| - 1 \tag{4.76}$$

$$\sum_{q \in \mathcal{Q}_u} \lambda^u_q = 1 \qquad \forall u \in V \tag{4.77}$$

$$x_{ij} \in \{0,1\} \qquad \forall ij \in E \tag{4.78}$$

$$\lambda^q_u \geq 0 \qquad \forall u \in U \quad \forall q \in \mathcal{Q}_u \tag{4.79}$$

Let $\pi^u_{ij} \leq 0$ be the dual prices of the constraints (4.75), and let $\mu_u \in \mathbb{R}$ be the dual prices of the convexity constraints (4.77) . There is one pricing problem for each commodity $u \in V$ asking for a negative-reduced cost spanning tree with flows:

$$\min \quad \sum_{ij \in A} c_{ij} \overline{f}^u_{ij} \sum_{ij \in A} w_r c_{ij} - \phi^r_{ij} \overline{f}^r_{ij} - \mu_r \tag{4.80}$$

s.t.

$$\sum_{uj} \overline{f}^u_{uj} = \sum_{h \in V \setminus \{u\}} w_{uh} \tag{4.81}$$

$$\sum_{hj \in A} \overline{f}^u_{hj} - \sum_{ih \in A} \overline{f}^u_{ih} = w_{uh} \qquad \forall h \in V \setminus \{u\} \tag{4.82}$$

$$\overline{f}^u_{ij} \leq M_u \overline{y}^u_{ij} \qquad \forall ij \in A \tag{4.83}$$

$$\sum_{ij \in A} \overline{y}^u_{ij} \leq |V| - 1 \tag{4.84}$$

$$\overline{y}^u_{ij} \in \{0,1\} \qquad \forall ij \in A \tag{4.85}$$

$$\overline{f}^u_{ij} \geq 0 \qquad \forall ij \in A \tag{4.86}$$

This problem is a fixed-cost network flow problem (FCNFP). The authors solve this pricing

problem with an heuristic algorithm. Besides that they also consider constraints to exclude nontree solutions.

4.3.3 Benders decomposition for the Path-Based formulation

Zetina, Contreras, Fernández and Luna [ZCFL19] proposed an algorithm that uses a strong Benders reformulation for the OCST within a branch-and-cut framework and obtain good solutions during the enumeration process. To apply Benders decomposition to the Path-Based formulation, the authors fix the variables x, leading to the following primal subproblem (PSP):

$$\min \quad \sum_{r\in R} w_r \sum_{ij\in A} c_{ij} y^r_{ij} \tag{4.87}$$

s.t.

$$\sum_{i,r_d\in A} y^r_{ir_d} = 1 \qquad \forall r \in R \tag{4.88}$$

$$\sum_{r_o,k\in A} y^r_{r_o k} = 1 \qquad \forall r \in R \tag{4.89}$$

$$\sum_{ij\in A} y^r_{ij} - \sum_{jk\in A} y^r_{jk} = 0 \qquad \forall r \in R \quad \forall j \in V \setminus \{r_o, r_d\} \tag{4.90}$$

$$y^r_{ij} + y^r_{ji} \leq x_{ij} \qquad \forall r \in R \quad \forall ij \in E \tag{4.91}$$

$$y^r_{ij} \geq 0 \qquad \forall r \in R \quad \forall ij \in A \tag{4.92}$$

Note that PSP can be split into $|R|$ independent shortest path problems PSP_r for each $r \in R$. Let λ be the dual variables of constraints (4.87), (4.88) , (4.90) and μ the dual variables of constraints (4.89). From strong duality, each PSP_r can be substituted by its dual LP , denoted by DSP_r , of the form:

$$\max \quad (\lambda^r_{r_d} - \lambda^r_{r_o}) - \sum_{ij\in A} \mu^r_{ij} x_{ij} \tag{4.93}$$

s.t.

$$\lambda^r_j - \lambda^r_i - \mu^r_{ij} \leq w_r(c_{ij}) \qquad \forall ij \in E \tag{4.94}$$

$$\lambda^r_i - \lambda^r_j - \mu^r_{ij} \leq w_r(c_{ij}) \qquad \forall ij \in E \tag{4.95}$$

$$\mu^r_{ij} \geq 0 \qquad \forall ij \in E \tag{4.96}$$

$$\lambda^r_i \in \mathbb{R} \qquad \forall i \in V \tag{4.97}$$

The set of extreme rays of DSP_r , obtained when it is unbounded, indexes the feasibility cuts of MP while the set of extreme points, obtained from the optimal solution of DSP_r , indexes the optimality cuts. The Master Problem (MP) is of the form:

$$\min \quad \sum_{r \in R} z_r \tag{4.98}$$

$s.t.$

$$z_r \geq \lambda^r_{r_d} - \lambda^r_{r_o} - \sum_{ij \in E} -\mu^r_{ij} x_{ij} \qquad \forall r \in R \quad (\lambda, \mu)_r \in \theta_r \tag{4.99}$$

$$z_r \geq \overline{\lambda}^r_{r_d} - \overline{\lambda}^r_{r_o} - \sum_{ij \in E} -\overline{\mu}^r_{ij} x_{ij} \qquad \forall r \in R \quad (\overline{\lambda}, \overline{\mu})_r \in \phi_r \tag{4.100}$$

$$\sum_{ij \in E} x_{ij} = |V| - 1 \tag{4.101}$$

$$x_{ij} \in \{0, 1\} \qquad \forall ij \in E \tag{4.102}$$

Where θ and ϕ represent the set of extreme points and extreme rays of DSP_r , respectively.

Chapter 5

Computational Experiments

This chapter is organized as follows. In Section 5.1, we describe the computational environment and the programs that we used to perform our experiments. In Section 5.2, we detail the implementation of the branch-and-cut algorithm using Gurobi optimizer. Section 5.3 describes how the OCST instances have been generated. Next, in Section 5.4, we compare the performance of the different MILP formulations for the OCST problem. Finally, Section 5.5 summarizes the results and the contributions of our experiments.

5.1 Computational environment and programs

To generate random instances for the OCST problem, we use the Python Networkx package [HSS08]. For the computational experiments, we used Gurobi optimizer [Gur19], a software package of optimization, as the solver for the MILP formulations.

The algorithms were implemented using the standard C++ programming language. To analyze the results, we use two popular Python data science libraries: Pandas and NumPy. Pandas library provides an API to manipulate data in a tabular format, while NumPy provides fast linear algebra operators [PTN+17].

The source code was implemented, compiled, and executed in a computer with 64 GB of memory RAM and a processor of 2.40 GHz. A more detailed description of the computer used to run the experiments is shown in Table 5.1.

Processor	
Model	Intel(R) Xeon(R) CPU E5620
Clock speed	2.40GHz
Cache L2	256KB
Cache L3	12 MB
RAM Memory	
Size	64 GB
Other	
SO	Debian GNU/Linux

Table 5.1: *Computational environment*

5.2 Separation algorithms and Gurobi parameters

In this section, we describe how we solved the separation problem for the OCST formulations. Besides, we describe how we configured the optimizer to test all formulations.

To address the separation problem, Gurobi generates and adds constraints (cuts) at runtime only as required (so-called lazy constraints). Gurobi implements lazy constraints by allowing user-defined code (callback function). This function is called periodically during the runtime of the solver. So, any solution that violates these constraints will be cut-off. Besides that, we configure Gurobi to use a single thread of the processor. Moreover, we deactivated all built-in cut generation options.

Each formulation for the OCST problem, in Chapter 4, considers x variables whose support induces a spanning tree, say T. We consider two cases to solve the separation algorithm.

- T is an undirected graph. This is the case of PB and FB formulations. In our implementations of these formulations, we consider two subcases.

  First case: when a feasible integer solution has been found. In this case, we consider the integral solution and implement a separation routine using a depth-first search (DFS). If T contains a cycle, say C, we add an inequality corresponding to the vertices of C.

  Second case: when in a node of the branch-and-cut algorithm, a noninteger feasible solution is found. To provide better lower bounds in the nodes of the branch-and-cut algorithm, we use the separation procedure that gives us a violated constraint if it exists (see Chapter 4).

- T is a rooted tree. In particular, the Rooted Tree formulation does not require any separation algorithm. However, we add SEC (an arborescence version) to obtain better lower bounds on the nodes of the branch-and-cut algorithm.

In the implementation of the branch-and-cut approach, to select variables to branch, different strategies are provided by Gurobi optimizer. The following options are available:

Pseudo Shadow Price Branching [Bea79], pseudo Reduced Cost Branching, Maximum Infeasibility Branching, and Strong Branching. In our work, we select the default values.

5.3 Description of the experiments

In our experiments, we generate random graphs using the Erdős-Rényi model. More specifically, we generate graphs $G = G(n, p)$, where n is the number of vertices, and p is the probability that an edge, say ij, exists in the graph. This is considered for each possible pair ij of vertices.

An instance (G, c, R, w) is generated in the following way. For fixed n and p, we generate a random graph $G = G(n, p)$. Then to generate the set R (of communication requirements) we fix a probability pR and generate a graph $G_R = G(n, pR)$. Then, we consider that $(u, v) \in R$ if, and only if, $uv \in E(G_R)$. Moreover, the values of the cost function c are randomly generated nonnegative values between 1 and 1000, and each requirement demand w_{ij}, is a random positive value between 1 and max$W$ (see the table below).

According to the value of p, we will have sparser or denser graphs. Also, pR let us control the size of the set R. The instances that are generated are then named as SparseRsmallW, SparseRlargeW, CompleteRsmallW, CompleteRlargeW, according to the values indicated in the table below.

In the tables on the computational results, we only mention the values of n and p and we name the DataSets according to the types of instances we have just mentioned.

As we stated earlier, we use the Python Networkx package [HSS08] to generate the instances. Since the input graphs must be connected, we only consider connected graphs (graphs that are not connected are discarded).

In the table below we show the names of the DataSets and the values of the parameters that we have considered.

DataSet	max$W$	pR	n	p
SparseRsmallW	1000	0.3	20, 40, 50, 60, 70	0.2, 0.5, 1.0
SparseRlargeW	1000000			
CompleteRsmallW	1000	1.0		
CompleteRlargeW	1000000			

Note that for each type of DataSet there are 15 different combinations, each corresponding to a pair (n, p), where $n \in \{20, 40, 50, 60, 70\}$ and $p \in \{0.2, 0.5, 1.0\}$. Moreover, for each such combination, we generate 20 instances. This gives a total of 300 instances for each DataSet type. The entries shown in the tables correspond to the average values considering these 20 instances.

5.4 Computational results

In this section we present the computational results regarding the implementation of the different formulations we have considered. Each instance was executed with a time limit of 1800 seconds.

For each formulation, we show a table containing the following information:

- n: the number of vertices of the graph G.
- p: the probability that an edge exists in G.
- Time: the running time (in seconds).
- Gap: the gap between lower and upper bounds. In the case of the OCST problem, a minimization problem, we calculate the gap using the following formula:

$$\text{gap} = (\text{UpperBound} - \text{LowerBound}) \ / \ \text{UpperBound}.$$

- Lazy Constraints: number of constraints added when the solver finds feasible integer solutions that do not satisfy constraints not added initially. That is the case of the Subtour Elimination Constraints.
- Cuts generated: number of inequalities, for fractional solutions, added to satisfy the Subtour Elimination Constraints. Implemented as user cuts, in Gurobi, they are useful for improving the lower bound in a node of the branch-and-cut algorithm.

In the second row, we show the functions that group the results. These are the maximum value, the minimum value, the mean, and the standard deviation. In the last column it is indicated the number of instances solved (to optimality).

In the following tables we show the results for DataSet CompleteRlargeW. We have run our experiments on other DataSets as well. In Table 5.6 we summarize the results obtained, showing only the number of intances solved.

n	p	Time				Gap		Lazy C.	Cuts	solved
		mean	std	max	min	mean	std	mean	mean	
20	0.2	8.4	16.4	71.5	0.3	0.0	0.0	0.0	3.4	20
20	0.5	13.7	21.4	76.4	0.6	0.0	0.0	0.0	3.7	20
20	1.0	27.0	60.2	270.6	1.6	0.0	0.0	0.0	3.0	20
40	0.2	1 180.5	694.1	1 801.3	28.0	0.0	0.1	0.0	17.2	8
40	0.5	1 421.4	597.4	1 802.8	165.0	0.0	0.0	0.0	12.4	8
40	1.0	1 292.3	699.6	1 800.6	66.8	0.1	0.1	0.0	7.2	8
50	0.2	1 700.0	315.0	1 803.5	437.5	0.1	0.1	0.0	5.7	2
50	0.5	1 675.4	344.7	1 811.4	505.9	0.1	0.1	0.0	2.7	2
50	1.0	1 807.8	15.0	1 858.5	1 800.5	0.2	0.2	0.0	1.6	0
60	0.2	1 805.1	11.7	1 848.6	1 800.2	0.5	0.4	0.0	0.8	0
60	0.5	1 795.6	27.2	1 814.7	1 680.6	0.9	0.3	0.0	0.2	0
60	1.0	1 899.1	110.2	2 201.7	1 801.9	1.0	0.2	0.0	0.0	0
70	0.2	1 801.3	2.1	1 810.3	1 800.7	1.0	0.0	0.0	0.0	0
70	0.5	1 976.4	180.8	2 297.2	1 801.6	1.0	0.0	0.0	0.0	0
70	1.0	3 631.6	2 466.6	12 046.2	1 820.4	1.0	0.0	0.0	0.0	0

Table 5.2: *Results for the Path-Based formulation on DataSet* CompleteRlargeW.

n	p	Time				Gap		Lazy C.	Cuts	solved
		mean	std	max	min	mean	std	mean	mean	
20	0.2	323.0	556.8	1 800.0	1.3	0.0	0.0	0.0	477.4	13
20	0.5	588.3	738.2	1 800.1	2.6	0.0	0.1	0.0	777.5	10
20	1.0	662.4	750.2	1 800.1	16.1	0.0	0.1	0.0	656.8	8
40	0.2	1 800.2	0.1	1 800.6	1 800.0	0.2	0.1	0.0	1 959.7	0
40	0.5	1 800.3	0.2	1 800.8	1 800.0	0.2	0.1	0.0	1 236.9	0
40	1.0	1 800.4	0.3	1 801.1	1 800.1	0.2	0.1	0.0	839.3	0
50	0.2	1 800.3	0.2	1 800.7	1 800.0	0.2	0.1	0.0	1 155.3	0
50	0.5	1 800.8	0.7	1 802.2	1 800.0	0.2	0.1	0.0	721.8	0
50	1.0	1 801.5	1.0	1 803.1	1 800.1	0.2	0.1	0.0	429.0	0
60	0.2	1 800.9	0.6	1 802.0	1 800.0	0.2	0.1	0.0	678.3	0
60	0.5	1 801.8	1.3	1 804.6	1 800.0	0.3	0.1	0.0	389.0	0
60	1.0	1 802.4	1.7	1 805.8	1 800.0	0.2	0.1	0.0	244.2	0
70	0.2	1 802.2	1.2	1 804.5	1 800.1	0.3	0.1	0.0	399.8	0
70	0.5	1 802.6	2.1	1 806.5	1 800.0	0.2	0.1	0.0	241.4	0
70	1.0	1 804.9	3.7	1 812.9	1 800.2	0.3	0.1	0.0	142.0	0

Table 5.3: *Results for the Relaxed Flow-Based formulation on DataSet* CompleteRLargeW.

n	p	Time				Gap		Lazy C.	Cuts	solved
		mean	std	max	min	mean	std	mean	mean	
20	0.2	417.7	630.3	1 800.0	5.6	0.0	0.0	0.0	488.0	13
20	0.5	674.7	765.8	1 800.1	6.9	0.0	0.1	0.0	853.9	10
20	1.0	765.0	742.6	1 800.1	30.0	0.0	0.1	0.0	678.1	8
40	0.2	1 800.2	0.2	1 800.6	1 800.0	0.2	0.1	0.0	1 790.4	0
40	0.5	1 800.4	0.3	1 800.9	1 800.0	0.2	0.1	0.0	1 076.8	0
40	1.0	1 800.6	0.4	1 801.4	1 800.0	0.2	0.1	0.0	652.0	0
50	0.2	1 800.4	0.4	1 801.2	1 800.0	0.2	0.1	0.0	1 061.6	0
50	0.5	1 800.6	0.6	1 801.9	1 800.0	0.2	0.1	0.0	615.6	0
50	1.0	1 800.9	0.8	1 802.9	1 800.0	0.3	0.1	0.0	329.1	0
60	0.2	1 801.1	0.8	1 802.6	1 800.0	0.2	0.1	0.0	611.4	0
60	0.5	1 801.9	1.2	1 803.4	1 800.2	0.3	0.1	0.0	321.0	0
60	1.0	1 802.5	2.0	1 806.1	1 800.0	0.3	0.1	0.0	150.1	0
70	0.2	1 801.0	1.3	1 804.2	1 800.0	0.3	0.1	0.0	361.7	0
70	0.5	1 802.6	1.9	1 806.4	1 800.1	0.3	0.1	0.0	159.5	0
70	1.0	1 804.0	4.8	1 814.1	1 800.1	0.3	0.1	0.0	25.0	0

Table 5.4: *Results for the Flow-Based formulation on DataSet* CompleteRLargeW.

n	p	Time				Gap		Lazy C.	Cuts	solved
		mean	std	max	min	mean	std	mean	mean	
20	0.2	1 800.1	0.1	1 800.3	1 800.0	0.2	0.1	0.0	128.6	0
20	0.5	1 800.1	0.1	1 800.3	1 800.0	0.2	0.1	0.0	241.1	0
20	1.0	1 800.1	0.1	1 800.3	1 800.0	0.2	0.1	0.0	242.7	0
40	0.2	1 801.7	1.3	1 803.8	1 800.0	0.3	0.1	0.0	43.5	0
40	0.5	1 802.2	1.2	1 803.7	1 800.0	0.3	0.1	0.0	40.3	0
40	1.0	1 801.8	1.7	1 804.4	1 800.0	0.3	0.1	0.0	24.4	0
50	0.2	1 804.1	2.7	1 809.0	1 800.1	0.3	0.1	0.0	21.5	0
50	0.5	1 805.6	3.0	1 810.6	1 800.1	0.3	0.1	0.0	13.7	0
50	1.0	1 806.1	3.4	1 812.4	1 800.0	0.3	0.1	0.0	4.7	0
60	0.2	1 809.0	6.3	1 819.7	1 800.1	0.3	0.1	0.0	11.5	0
60	0.5	1 809.8	5.9	1 820.0	1 800.1	0.4	0.1	0.0	8.0	0
60	1.0	1 813.2	7.4	1 825.4	1 800.6	0.3	0.1	0.0	6.2	0
70	0.2	1 820.9	11.8	1 838.7	1 800.2	0.4	0.1	0.0	7.1	0
70	0.5	1 818.0	12.5	1 842.8	1 800.5	0.3	0.1	0.0	4.6	0
70	1.0	1 821.1	14.3	1 845.3	1 801.4	0.3	0.1	0.0	0.2	0

Table 5.5: *Results for the Rooted Tree based formulation on DataSet* CompleteRLargeW.

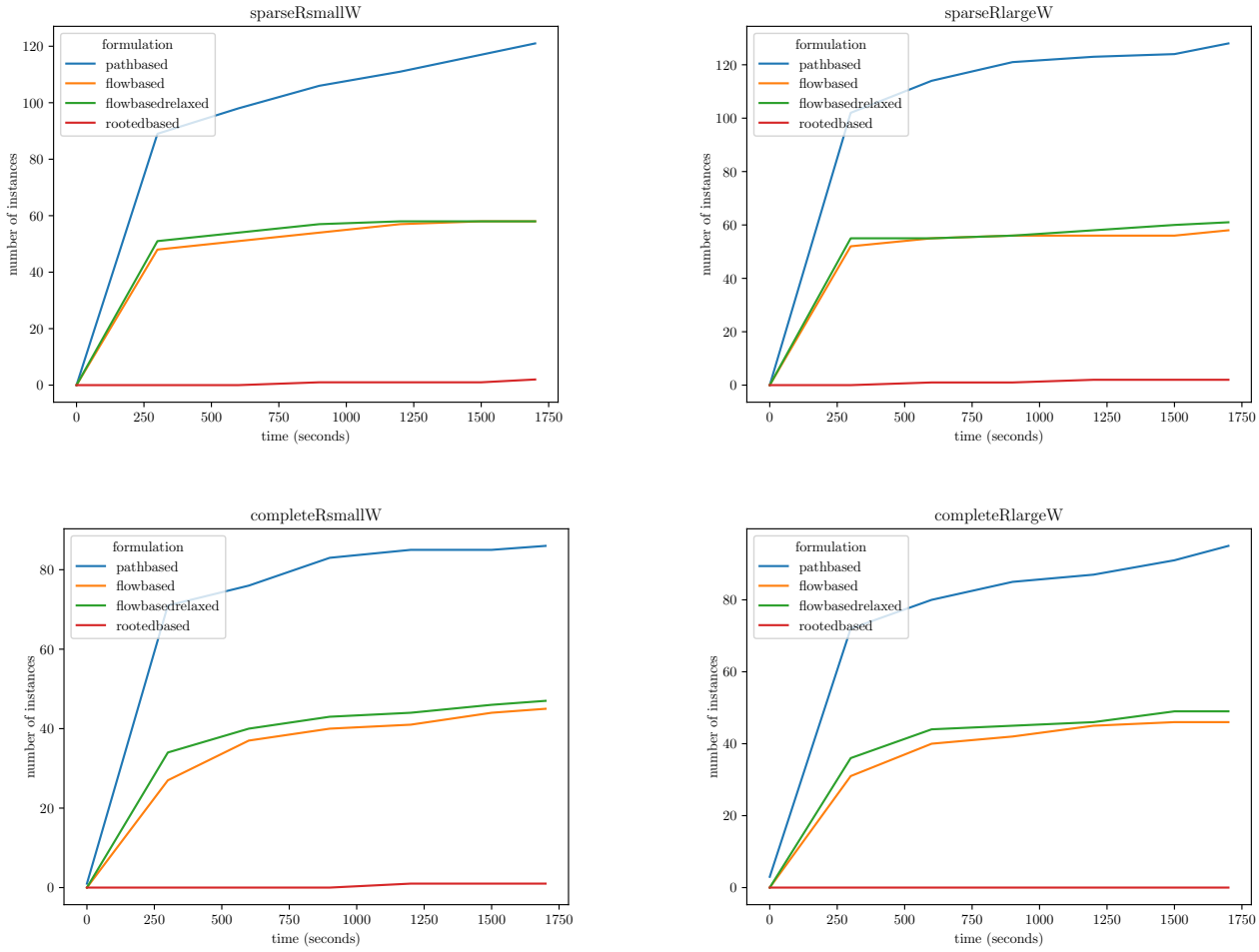


Table 5.6: *Number of instances solved on different DataSets.*

5.4.1 Instances solved and instances with tight gaps

In this subsection, for each DataSet type we show the number of instances solved to optimality (Figure 5.1) and the number of instances not solved, but for which tight gaps were obtained (Figure 5.2) within the time limit of 1800 seconds. We consider four different formulations, each one represented in a different color.

The reader interested in the implementations, instances, and results can consult the public repository:

https://github.com/jainor/ocst-problem/.

5.5 Concluding remarks

The tables with the computational results shown in the previous section indicate that the Path-Based formulation has the best performance among the four formulations that we have implemented. These results are as expected in the literature. This formulation gives

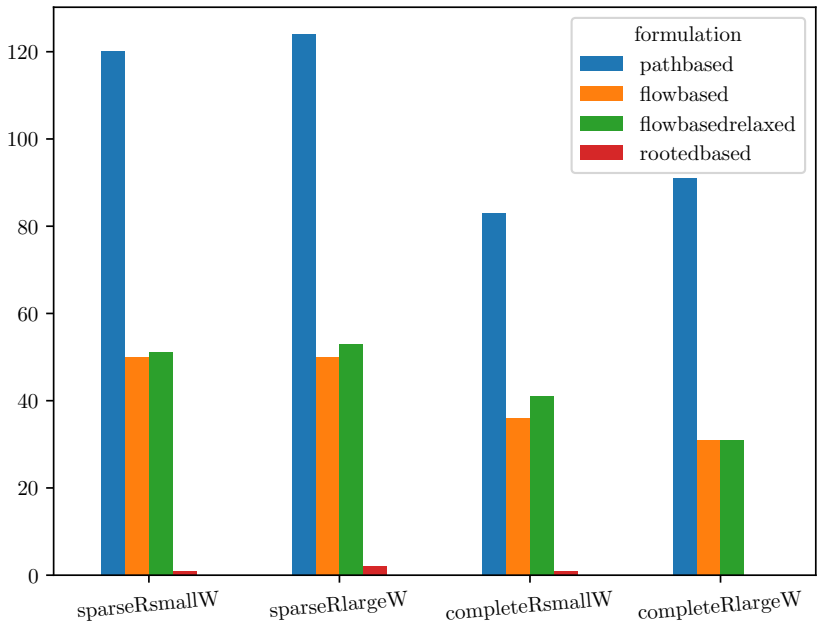


Figure 5.1: *Number of instances solved per DataSet.*

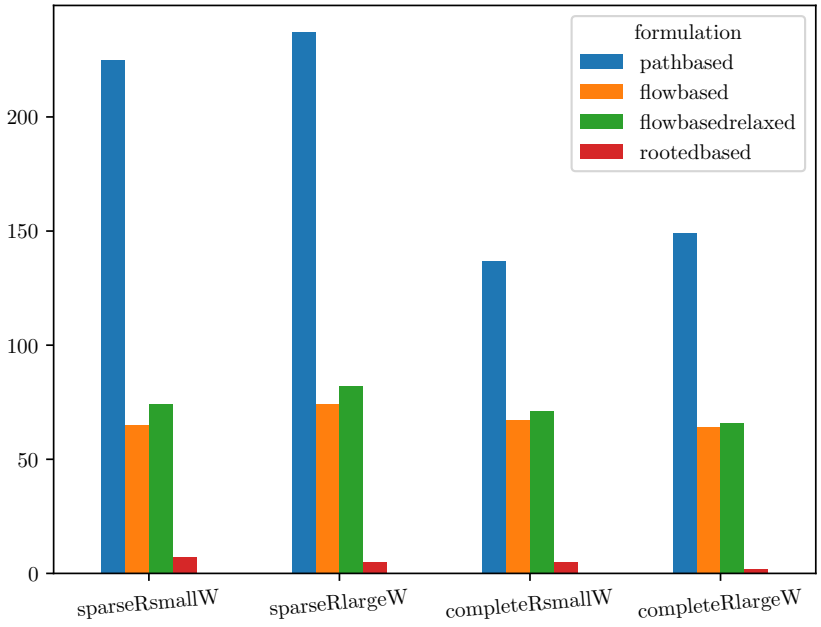


Figure 5.2: *Number of instances per DataSet with a gap less than or equal to 0.1.*

tight lower bounds and is efficient for instances up to 30 nodes. The Rooted Tree formulation leads to the worst performance in terms of efficiency (gaps) and runtime. The Flow-Based and Relaxed Flow-Based formulations have similar performance, with a slight advantage for the relaxed version.

We observe that we have made no additional assumption on the instances (they were all randomly generated). Perhaps, on some more special instances (metric cases or real datas) these implementations could perform better.

The implementations we have carried out were aimed at gaining more experience with the branch-and-cut technique and the use of Gurobi optimizer. We did not expect that these implementations would outperform other implementations of this nature. Many other enhancements are needed to be able to solve to optimality instances of medium size (from 40 to 50 vertices).

Current column generation techniques for the OCST problem use the Flow-Based formulation. Although the Relaxed Flow-Based formulation has shown a slight advantage compared to the Flow-Based formulation, possibly this gain may not make much difference when a column generation approach is used.

As we have mentioned in Chapter 3, the algorithm with best performance among all the exact algorithms that have been proposed for the OCST problem has been developed by Zetina, Contreras, Fernández and Luna [ZCFL19]. It considers an arc-based MILP formulation and is based on Benders decomposition (integrated within a branch-and-bound framework). It expanded the limits of solvability for the OCST problem from 40 to 60 vertices.

We hope the present work can be used as a starting point for other implementations, possibly using stronger valid inequalities and good separations routines.

Chapter 6

Final considerations

In this thesis, we addressed the Optimal Communication Spanning Tree (OCST) problem, an interesting combinatorial optimization problem, known to be NP-hard.

As many concepts of different areas (graph theory, network flow, integer linear programming, approximation algorithms) are needed to understand the results mentioned or proved here, to make this text self-contained, these concepts were presented in Chapter 2. Then, in Chapter 3 we formally defined the OCST problem, presented some variations and special cases, and mentioned some applications. We also presented an overview on this problem, mentioning the main results we have found in the literature. In this chapter, we focused on two special cases of the OCST problem. This was a very enriching experience, as it required a lot of reading and learning concepts of different areas.

In this work, we mainly focused our discussion on exact algorithms. First, we investigated special cases of the OCST problem for which polynomial-time algorithms have been developed. In particular, we found very interesting the result concerning the 1-source OCST problem and its relation with the min-cost flow problem. However, this part was not included in this work. The study of the Gomory-Hu tree was specially rewarding. This was done in Chapter 3.

Then, in Chapter 4 we addressed the main mixed integer linear programming models that have been proposed for the OCST problem. For that, we first presented some ILP and MILP formulations for the spanning tree problem. This way, we revised some classical results on this topic, including a result on the separation of a well-known class of inequalities – the Subtour Elimination Constraints – interesting on its own right, considering that they are used in many problems.

In Chapter 5 we presented our computational experiments with the implementation of some MILP formulations for the OCST problem using a branch-and-cut approach. The source code of these implementations will be made available to anyone interested in testing a new formulation or tighten an existing formulation with new inequalities.

For further investigations on this topic, perhaps the Relaxed Flow-Based formulation is

a good candidate. A challenging work, in our view, would be to develop an approach using row and column generation algorithms like Dantzig-Wolfe decomposition. This approach would lead to a pricing problem similar to the 1-source OCST problem. To the best of our knowledge about the current state of the art –on row and column generations–, the Flow-Based formulation is used with relaxation only in its pricing subproblem. We need a deeper understanding of this approach to accomplish this task, but it seems that this would be a possible direction to explore.

We did not focus on approximation algorithms for the OCST problem and some variants, but it seems to be another line of research that would be interesting to explore. In Chapter 3 we only mentioned some main results, but this short survey may be useful to someone interested in this line of research.

In conclusion, we think the study of this topic brought us the opportunity to learn many different results and approaches. We still feel that there is much more to learn, and that we have only scratched the first layers, and we have to explore many more layers to be able to contribute with original and relevant results. This work has given me encouragement to further explore the problems mentioned here and/or other related problems.

Bibliography

[AH73] Donald L. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM J. Appl. Math.*, 25:403–423, 1973. 40

[AM87] R. K. Ahuja and V. V. S. Murty. Exact and heuristic algorithms for the optimum communication spanning tree problem. *Transportation Sci.*, 21(3):163–170, 1987. 29

[AMS84] Sunita Agarwal, A. K. Mittal and Prabha Sharma. Constrained optimum communication trees and sensitivity analysis. *SIAM J. Comput.*, 13(2):315–328, 1984. 29

[AV19] Yogesh Kumar Agarwal and Prahalad Venkateshan. New valid inequalities for the optimal communication spanning tree problem. *INFORMS J. Comput.*, 31(2):268–284, 2019. 31

[Bar96] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science (Burlington, VT, 1996)*, pages 184–193. IEEE Comput. Soc. Press, Los Alamitos, CA, 1996. 30

[Bar98] Yair Bartal. On approximating arbitrary metrices by tree metrics. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 161–168. ACM, 1998. 30

[Bea79] Evelyn Beale. Branch and bound methods for mathematical programming systems. volume 5, pages 201–219. 1979. Discrete optimization (Proc. Adv. Res. Inst. Discrete Optimization and Systems Appl., Banff, Alta., 1977), II. 69

[BHR12] Binh-Minh Bui-Xuan, Michel Habib and Michaël Rao. Tree-representation of set families and applications to combinatorial decompositions. *European J. Combin.*, 33(5):688–711, 2012. 35

[BJN+98] Cynthia Barnhart, Ellis L. Johnson, George L. Nemhauser, Martin W. P. Savelsbergh and Pamela H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.*, 46(3):316–329, 1998. 25

[BM08] J. Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008. 18

[CCPS98] William J. Cook, William H. Cunningham, William R. Pulleyblank and Alexander Schrijver. *Combinatorial optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York, 1998. A Wiley-Interscience Publication. 45

[CCZ14] Michele Conforti, Gérard Cornuéjols and Giacomo Zambelli. *Integer programming*, volume 271 of *Graduate Texts in Mathematics*. Springer, Cham, 2014. 21, 24

[CFM09] Iván A. Contreras, Elena Fernández and Alfredo Marín. Tight bounds from a path based formulation for the tree of hub location problem. *Comput. Oper. Res.*, 36(12):3117–3127, 2009. 29

[CFM10a] Iván A. Contreras, Elena Fernández and Alfredo Marín. Lagrangean bounds for the optimum communication spanning tree problem. *TOP*, 18(1):140–157, 2010. 31, 51

[CFM10b] Iván A. Contreras, Elena Fernández and Alfredo Marín. The tree of hubs location problem. *European J. Oper. Res.*, 202(2):390–400, 2010. 32, 34

[Dan63] George B. Dantzig. *Linear programming and extensions*. Princeton University Press, Princeton, N.J., 1963. 18

[DF56] George B. Dantzig and Delbert Ray Fulkerson. On the max-flow min-cut theorem of networks. In *Linear inequalities and related systems*, Annals of Mathematics Studies, no. 38, pages 215–221. Princeton University Press, Princeton, N. J., 1956. 23

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. 18

[Dij59] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1:269–271, 1959. 34, 63

[Din06] Yefim Dinitz. Dinitz' algorithm: The original version and even's version. In Oded Goldreich, Arnold L. Rosenberg and Alan L. Selman, editors, *Theoretical Computer Science, Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 218–240. Springer, 2006. 41

[Edm71] Jack R. Edmonds. Matroids and the greedy algorithm. *Math. Programming*, 1:127–136, 1971. 49

[FF57] L. R. Ford, Jr. and D. R. Fulkerson. A simple algorithm for finding maximal network flows and an application to the Hitchcock problem. *Canadian J. Math.*, 9:210–218, 1957. 41

[FLH+13] Elena Fernández, Carlos Luna-Mota, Achim Hildenbrandt, Gerhard Reinelt and Stefan Wiesberg. A flow formulation for the optimum communication spanning tree. *Electron. Notes Discret. Math.*, 41:85–92, 2013. 29, 31, 53

[FLS02] Matteo Fischetti, Giuseppe Lancia and Paolo Serafini. Exact algorithms for minimum routing cost trees. *Networks*, 39(3):161–173, 2002. 31

[FM07] Thomas Fischer and Peter Merz. A memetic algorithm for the optimum communication spanning tree problem. In Thomas Bartz-Beielstein, María J. Blesa Aguilera, Christian Blum, Boris Naujoks, Andrea Roli, Günter Rudolph and Michael Sampels, editors, *Hybrid Metaheuristics, 4th International Workshop, HM 2007, Dortmund, Germany, October 8-9, 2007, Proceedings*, volume 4771 of *Lecture Notes in Computer Science*, pages 170–184. Springer, 2007. 29

[GH61] Ralph E. Gomory and T. C. Hu. Multi-terminal network flows. *J. Soc. Indust. Appl. Math.*, 9:551–570, 1961. 35, 36

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 19

[GLS88] Martin Grötschel, László Lovász and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*, volume 2 of *Algorithms and Combinatorics: Study and Research Texts*. Springer-Verlag, Berlin, 1988. 21

[GT88] Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *J. Assoc. Comput. Mach.*, 35(4):921–940, 1988. 41

[Gur19] LLC. Gurobi, Gurobi Optimization. Gurobi optimizer reference manual, 2019. 67

[Hač79] Leonid Hačijan. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244(5):1093–1096, 1979. 18

[HLN10] Anh Tuan Hoang, Vinh Trong Le and Gia Nhu Nguyen. A novel particle swarm optimization-based algorithm for the optimal communication spanning tree problem. In *2010 Second International Conference on Communication Software and Networks*, pages 232–236, 2010. 29

[HSS08] Aric Hagberg, Pieter J. Swart and Daniel Schult. Exploring network structure, dynamics, and function using networkx. Relatório técnico, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008. 67, 69

[Hu.74] T. C. Hu. Optimum communication spanning trees. *SIAM J. Comput.*, 3:188–195, 1974. 29, 34, 36, 42

[JLK78] David S. Johnson, Jan Karel Lenstra and A. H. G. Rinnooy Kan. The complexity of the network design problem. *Networks*, 8(4):279–285, 1978. 29, 30, 42

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984. 18

[KMR97] David R. Karger, Rajeev Motwani and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18(1):82–98, 1997. 60

[Kru56] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, 7:48–50, 1956. 28

[LRS11] Lap Chi Lau, R. Ravi and Mohit Singh. *Iterative methods in combinatorial optimization*. Cambridge Texts in Applied Mathematics. Cambridge University Press, New York, 2011. 49

[Lun16] Carlos Luna-Mota. *The Optimum Communication Spanning Tree Problem: properties, models and algorithms*. Tese de Doutorado, UPC, Departament d'Estadística i Investigació Operativa, 2016. 29, 31, 52, 56, 58

[Mey74] Robert R. Meyer. On the existence of optimal solutions to integer and mixed-integer programming problems. *Math. Programming*, 7:223–235, 1974. 19

[MJSS16] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe and Edward C. Sewell. Branch-and-bound algorithms: a survey of recent advances in searching, branching, and pruning. *Discrete Optim.*, 19:79–102, 2016. 25

[MW84] Thomas L. Magnanti and Richard T. Wong. Network design and transportation planning: Models and algorithms. *Transp. Sci.*, 18(1):1–55, 1984. 13

[MW95] Thomas L. Magnanti and Laurence A. Wolsey. Optimal trees. In *Network models*, volume 7 of *Handbooks Oper. Res. Management Sci.*, pages 503–615. North-Holland, Amsterdam, 1995. 45

[NLL13] Gia Nhu Nguyen, Dac-Nhuong Le and Dang Nguyen Le. A novel ant colony optimization-based algorithm for the optimal communication spanning tree problem. *International Journal of Computer Theory and Engineering*, 5(3):509, 2013. 29

[NN12] Jaroslav Nešetřil and Helena Nešetřilová. The origins of minimal spanning tree algorithms—Borůvka and Jarník. In *Doc. Math.*, Extra vol.: Optimization stories, pages 127–141. 2012. 28

[NQN11] Minh Hieu Nguyen, PhanTan Quoc and Nguyen Duc Nghia. An approach of ant algorithm for solving minimum routing cost spanning tree problem. In Huynh Quyet Thang and Dinh Khang Tran, editors, *Proceedings of the 2011 Symposium on Information and Communication Technology, SoICT 2011, Hanoi, Viet Nam, October 13-14, 2011*, pages 5–10. ACM, 2011. 29

[PK94] Charles C. Palmer and Aaron Kershenbaum. Two algorithms for finding optimal communications spanning trees. Relatório técnico, Thomas J. IBM Research Center – Research Division, 1994. 29

[PK95] Charles C. Palmer and Aaron Kershenbaum. An approach to a problem in network design using genetic algorithms. *Networks*, 26(3):151–163, 1995. 29

[PR98] David Peleg and Eilon Reshef. Deterministic polylog approximation for minimum communication spanning trees. In Kim Guldstrand Larsen, Sven Skyum and Glynn Winskel, editors, *Automata, Languages and Programming, 25th International Colloquium, ICALP'98, Aalborg, Denmark, July 13-17, 1998, Proceedings*, volume 1443 of *Lecture Notes in Computer Science*, pages 670–681. Springer, 1998. 30

[PTN+17] Shoumik Palkar, James Thomas, Deepak Narayanan, Anil Shanbhag, Rahul Palamuttam, Holger Pirk, Malte Schwarzkopf, Saman Amarasinghe, Samuel Madden and Matei Zaharia. Weld: Rethinking the interface between data-intensive applications. *arXiv preprint arXiv:1709.06416*, 2017. 67

[RF15] Santiago V. Ravelo and Carlos Eduardo Ferreira. PTAS's for some metric p-source communication spanning tree problems. In *WALCOM: algorithms and computation*, volume 8973 of *Lecture Notes in Comput. Sci.*, pages 137–148. Springer, Cham, 2015. 31

[RF17] Santiago V. Ravelo and Carlos Eduardo Ferreira. A PTAS for the metric case of the minimum sum-requirement communication spanning tree problem. *Discrete Appl. Math.*, 228:158–175, 2017. 31

[Sch03] Alexander Schrijver. *Combinatorial optimization. Polyhedra and efficiency. Vol. A*, volume 24 of *Algorithms and Combinatorics*. Springer-Verlag, Berlin, 2003. Paths, flows, matchings, Chapters 1–38. 21

[SK83] David Sankoff and Joseph B. Kruskal, editors. *Time warps, string edits, and macromolecules: the theory and practice of sequence comparison*. Addison-Wesley Publishing Company, Advanced Book Program, Reading, MA, 1983. 32

[Soa06] Sang-Moon Soak. A new evolutionary approach for the optimal communication spanning tree problem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 89(10):2882–2893, 2006. 29

[Tan12] Quoc Phan Tan. A genetic approach for solving minimum routing cost spanning tree problem. *International Journal of Machine Learning and Computing*, 2(4):410, 2012. 29

[TI18] Christian Tilk and Stefan Irnich. Combined column-and-row-generation for the optimal communication spanning tree problem. *Comput. Oper. Res.*, 93:113–122, 2018. 29, 31, 62, 63

[Wat95] Michael S. Waterman. *Introduction to computational biology - maps, sequences, and genomes: interdisciplinary statistics*. CRC Press, 1995. 32

[WC04] Bang Ye Wu and Kun-Mao Chao. *Spanning trees and optimization problems*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2004. 29, 30

[WCT00a] Bang Ye Wu, Kun-Mao Chao and Chuan Yi Tang. Approximation algorithms for some optimum communication spanning tree problems. *Discrete Appl. Math.*, 102(3):245–266, 2000. 30, 31

[WCT00b] Bang Ye Wu, Kun-Mao Chao and Chuan Yi Tang. Approximation algorithms for the shortest total path length spanning tree problem. *Discrete Appl. Math.*, 105(1-3):273–289, 2000. 30

[WCT00c] Bang Ye Wu, Kun-Mao Chao and Chuan Yi Tang. A polynomial time approximation scheme for optimal product-requirement communication spanning trees. *J. Algorithms*, 36(2):182–204, 2000. 30, 31

[WJ94] Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *J. Comput. Biol.*, 1(4):337–348, 1994. 32

[WLB+00] Bang Ye Wu, Giuseppe Lancia, Vineet Bafna, Kun-Mao Chao, R. Ravi and Chuan Yi Tang. A polynomial-time approximation scheme for minimum routing cost spanning trees. *SIAM J. Comput.*, 29(3):761–778, 2000. 13, 30, 31, 32

[Wol98] Laurence A. Wolsey. *Integer programming*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, Inc., New York, 1998. A Wiley-Interscience Publication. 25

[Won80] Richard T. Wong. Worst-case analysis of network design problem heuristics. *SIAM J. Algebraic Discrete Methods*, 1(1):51–63, 1980. 30

[WS11] David P. Williamson and David B. Shmoys. *The design of approximation algorithms*. Cambridge University Press, Cambridge, 2011. 26

[Wu02] Bang Ye Wu. A polynomial time approximation scheme for the two-source minimum routing cost spanning trees. *J. Algorithms*, 44(2):359–378, 2002. 30, 31, 34

[Wu04] Bang Ye Wu. Approximation algorithms for the optimal p-source communication spanning tree. *Discrete Appl. Math.*, 143(1-3):31–42, 2004. 30, 31

[ZCFL19] Carlos Armando Zetina, Iván A. Contreras, Elena Fernández and Carlos Luna-Mota. Solving the optimum communication spanning tree problem. *European J. Oper. Res.*, 273(1):108–117, 2019. 29, 31, 62, 65, 75