Università degli Studi di Salerno

dipartimento di informatica

Dottorato di Ricerca in Informatica

ciclo xiv

Ph.D. Thesis in Computer Science

Models and Algorithms for Some Covering Problems on Graphs

Candidate

Selene Silvestri

Tutor

Prof. Raffaele Cerulli

Co-Tutor

Prof. Gilbert Laporte

Coordinator Prof. Gennaro Costagliola

2014/2015

A mia madre, mio padre, mio fratello.. a me.

Acknowledgements

Il primo grazie va al mio tutor, il Prof. Raffaele Cerulli, per avermi introdotta alla Ricerca Operativa, per aver creduto nelle mie capacità (anche prima di me) e per aver guidato i miei passi in tutti questi anni.

My first thanks goes to my tutor, Prof. Raffaele Cerulli, for introducing me to the Operational Research, for believing in my potential (even before I did) and for having guided my steps all these years.

Ringrazio tantissimo il Prof. Giovanni Rinaldi del CNR di Roma per i suoi insegnamenti e la grande disponibilità. È stato un onore poter lavorare con lui.

Many thanks to Prof. Giovanni Rinaldi from the CNR of Rome, for his teachings and his great helpfulness. It has been an honor working with him.

Un grazie di cuore a tutti i componenti del mio gruppo dell'Università di Salerno. In particolare, grazie al dott. Carmine Cerrone, per essere stato un buon amico e un esempio da seguire e al dott. Francesco Carrabs, per avermi aiutata ogni volta che ne ho avuto bisogno.

A heartfelt thanks to all members of my group at the University of Salerno. In particular, thanks to Dr. Carmine Cerrone, for being a good friend and an example to follow, and to Dr. Francesco Carrabs, for helping me whenever I needed.

Ringrazio con affetto Vinicius, Xiaolu, Yunfei, Aurélien, Slavic, Andrea, Giusy e tutti gli amici

I affectionately thank Vinicius, Xiaolu, Yunfei, Aurélien, Slavic, Andrea, Giusy and all friends of *del Cirrelt di Montréal. È stato un bellissimo periodo!*

Grazie a Vittorio e a tutti gli amici per esserci stati.

Un grazie speciale va poi ai miei genitori, a mio fratello Ivan e ad Antonio che mi hanno aiutata e incoraggiata in ogni momento.

L'ultimo grande grazie va al mio co-tutor, il Prof. Gilbert Laporte dell'HEC di Montréal, al quale sono grata per il tempo che mi ha dedicato, per la sua pazienza, per il prezioso aiuto. Sinceramente grazie!

the Cirrelt of Montréal. I really had a good time!

Thanks to Vittorio and to all my friends for having been there.

A special thanks goes to my parents, to my brother Ivan and to Antonio for the encouragement and help.

My last big thanks goes to my co-tutor, Prof. Gilbert Laporte from the HEC of Montréal, I am extremely grateful to him for the time he has dedicated to me, for his patience, for the precious help. Sincerely thanks!

May 10, 2016

Selene Silvestri

Contents

List of Figures **7**

List of Tables **9**

Introduction **11**

1 General concepts: a brief overview **14**

1.1 Integer and Combinatorial Optimization 14

1.2 Computational complexity 15

1.3 Graph theory 16

1.4 Polyhedral theory 19

1.5 Branch-and-Cut Algorithm 23

2 Spanning Tree and Cycle Cover **26**

2.1 Introduction 26

2.2 Spanning Trees 26

2.2.1 Spanning Tree on edge-labeled graphs 29

2.3 Cycle Cover Overview 30

2.3.1 Cycle Cover on edge-labeled graphs 33

3 The Rainbow Cycle Cover Problem **34**

3.1 Introduction and Problem Description 34

3.2 Mathematical formulation and Properties 35

3.2.1 Properties of a Rainbow Cycle Cover 37

3.3 Valid inequalities 40

3.4 Branch-and-cut algorithm 45

3.5 Computational results 46
3.5.1 LP lower bounds and duality gaps 48

4 The Rainbow Spanning Forest Problem 53
4.1 Introduction and Problem Description 53
4.2 Problem Complexity 54
4.2.1 Problem Complexity: A Polynomial Case 59
4.3 Mathematical Formulations 60
4.3.1 First Mathematical Formulation 60
4.3.2 Second Mathematical Formulation 62
4.3.2.1 Valid inequalities for ILP2 65
4.4 The Greedy Algorithm 68
4.4.1 The multi-start scheme 74
4.5 Computational results 75

5 A Branch-and-Cut Algorithm for the Minimum Branch Vertices Spanning Tree Problem 82
5.1 Introduction 82
5.2 Undirected formulation, properties and bounds 84
5.2.1 Spanning tree properties 85
5.3 Polyhedral analysis of the undirected formulation 88
5.4 Directed and hybrid reformulations 97
5.5 Branch-and-cut algorithm 100
5.6 Computational results 102
5.6.1 LP lower bounds and duality gaps 106

Conclusions and Future Works 109

References 112

Nomenclature 117

List of Figures

1.1 Map of Königsberg in Euler's time. 17

1.2 Euler's representation of the bridges of Königsberg. 17

1.3 The black dots represent the set of integral points S. On the left a set containing S, on the right $conv(S)$. 19

1.4 Linear system defining $S \subset \mathbb{R}^2$. 22

1.5 Branch-and-Cut Algorithm. 24

2.1 The Minimum Spanning Tree: an example. 27

2.2 The Traveling Salesman Problem: an example. 30

3.1 Solution where $\gamma_1 = 1$ and $\gamma_2 = 0.34$ and constraints (3.24) are violated by v_4^2 and v_5^2 in $c = 2$. 41

3.2 A constraint (3.33) is violated for $k = k_2$ 44

3.3 An example of valid inequalities (3.36) 45

4.1 (a) A generic instance of the Minimum Set Covering Problem, and, (b) the corresponding instance of the Bounded Rainbow Spanning Problem. 55

4.2 (a) A generic instance of the 3-SAT Problem, and, (b) the corresponding instance of the Bounded Rainbow Spanning Forest Problem in Edge-Colored Trees. . . 58

4.3 Non-integer solution. 65

4.4 (a) An example of valid inequalities 4.25, and, (b) an example of valid inequalities 4.26. 68

4.5 (a) An example of valid inequalities 4.27, and, (b) an example of valid inequalities 4.28. 69

4.6 Feasible rainbow spanning forest F_k with four components 71

4.7 Feasible solution F_{k+1} obtained by adding edge (v_3, v_8). 72
4.8 Feasible solution F_{k+1} obtained by adding edge (v_3, v_{11}). 73
4.9 *The AvgGap of GA and MS algorithms on the larger scenarios.* 79

5.1 For a given graph on the left, two spanning trees with one and two branch vertices. 83
5.2 For a given graph on the left, v is a cut vertex u is not a cut vertex. 88

List of Tables

3.1 Summary of computational results for the RCCP 49
3.2 Linear programming lower bounds for RCCP 50
3.3 Linear programming duality gap for RCCP 51

4.1 Test results of ILP1 model, GA and MS algorithms on the small scenarios. 77
4.2 Test results of GA and MS on the large scenarios. 81

5.1 Undirected formulation: computational results for small instances . 103
5.2 Undirected formulation: computational results for medium instances 104
5.3 Hybrid formulation: computational results for small and medium instances 104
5.4 Hybrid formulation: computational results for large instances 105
5.5 Hybrid formulation: lower bounds for MBVP on small and medium instances 106
5.6 Hybrid formulation: lower bounds for MBVP on large instances . . 107
5.7 Hybrid formulation: duality gap on small and medium instances . . 107
5.8 Hybrid formulation: duality gap on large instances 108

“*If we knew what it was we were doing, it would not be called research, would it?*”

Albert Einstein

“*The scientist is not a person who gives the right answers, he is one who asks the right questions.*”

Claude Levi-Strauss

“*Somewhere, something incredible is waiting to be known.*”

Carl Sagan

..and that day I would like to be there!

Introduction

Several real-life problems as well as problems of theoretical importance within the field of Operations Research are combinatorial in nature. *Combinatorial Optimization* deals with decision-making problems defined on a discrete space. Out of a finite or countably infinite set of feasible solutions, one has to choose the best one according to an objective function. Many of these problems can be modeled on undirected or directed graphs. Some of the most important problems studied in this area include the *Minimum Spanning Tree Problem*, the *Traveling Salesman Problem*, the *Vehicle Routing Problem*, the *Matching Problem*, the *Maximum Flow Problem.* Some combinatorial optimization problems have been modeled on colored (labeled) graphs. The colors can be associated to the vertices as well as to the edges of the graph, depending on the problem. The *Minimum Labeling Spanning Tree Problem* and the *Minimum Labeling Hamiltonian Cycle Problem* are two examples of problems defined on edge-colored graphs.

Combinatorial optimization problems can be divided into two groups, according to their complexity. The problems that are *easy* to solve, i.e. problems polynomially solvable, and those that are *hard*, i.e. for which no polynomial time algorithm exists. Many of the well-known combinatorial optimization problems defined on graphs are hard problems in general. However, if we know more about the structure of the graph, the problems can become more tractable. In some cases, they can even be shown to be polynomial-time solvable. This particularly holds for trees.

In the last 80 years combinatorial optimization problems have been addressed through various modeling and algorithmic approaches, and many papers proposing algorithms which solve them efficiently have been published. These algorithmic approaches to combinatorial optimization problems can be classified as either *exact* or *heuristic.* Exact techniques are applied in order to find optimal solutions to the problems. Because of the complexity of the problems, exact approaches are not able to prove the optimality for *large instances*. The concept of large instance is related to the problem under consideration. When an exact approach fails, heuristic techniques are usually applied. The aim of a heuristic is to produce good feasible solutions within a reasonable time. Heuristics can also be embedded within exact approaches, e.g. they may be used to generate good initial feasible solutions.

This dissertation is devoted to the study of three different problems of combinatorial optimization defined on graphs and belonging to the class of Spanning Tree and Cycle Cover Problems: the *Rainbow Cycle Cover Problem* (RCCP), the *Rainbow Spanning Forest Problem* (RSFP) and the *Minimum Branch Vertices Spanning Tree Problem* (MBVP).

Given a connected and undirected graph $G = (V, E, L)$ and a coloring function ℓ that assigns a color to each edge of G from the finite color set L, a cycle whose edges have all different colors is called a rainbow cycle. The RCCP consists of finding the minimum number of disjoint rainbow cycles covering G. The RCCP on general graphs is known to be $\mathcal{NP}$-hard.

Given a graph $G = (V, E, L)$ and a coloring function $\ell : E \to L$ that assigns a color to each edge of G from a finite color set L, the Rainbow Spanning Forest Problem (RSFP) consists of finding a spanning forest of G such that the number of rainbow components is minimum. A component of the forest whose edges have all different colors is called rainbow component. The RSFP on general graphs is known to be $\mathcal{NP}$-hard.

Given a connected undirected graph $G = (V, E)$, the Minimum Branch Vertices Spanning Tree Problem (MBVP) asks for a spanning tree of G with the minimum number of vertices having degree greater than two in the tree. These are called *branch vertices*. This problem is known to be $\mathcal{NP}$-hard.

Chapter 1 provides a brief overview on some basic concepts from *integer and combinatorial optimization*, *graph theory*, *computational complexity* and *polyhedral analysis*. Moreover we present a brief description of the *Branch-and-Cut Algorithm*.

In Chapter 2 we present some Spanning Tree and Cycle Cover Problems that are standard topics in the combinatorial optimization literature and are frequently embedded as subproblems in more general problems. The two best-known and studied problems are probably the *Minimum Spanning Tree Problem* and the *Traveling Salesman Problem*.

In Chapter 3 we model and solve the *Rainbow Cycle Cover Problem* (RCCP). We present an integer linear mathematical formulation and we describe some properties that a rainbow cycle cover must satisfy. Moreover we derive valid inequalities for the RCCP and we solve it by branch-and-cut. Computational results are reported on randomly generated instances.

Chapter 4 is devoted to the *Rainbow Spanning Forest Problem* (RSFP). We prove that the problem is $\mathcal{NP}$-hard on trees and we provide a polynomial case. Moreover we propose two new integer mathematical formulations, (ILP1) and (ILP2), and for the second one we propose some valid inequalities. To solve large instances we present a greedy algorithm and a multi-start scheme applied to the greedy algorithm to improve its results. We show the computational results obtained by solving the ILP1, the greedy algorithm and the multi-start scheme.

In Chapter 5 we model the *Minimum Branch Vertices Spanning Tree Problem* (MBVP) as an integer linear program, with undirected variables and we also investigate some properties of the problem and the LP relaxation. Moreover, we derive the dimension of the polyhedron of integer solutions as well as some valid inequalities and prove than some these are facet defining. We then develop a hybrid formulation containing undirected and directed variables. Both model are solved by branch-and-cut. Comparative computational results show the superiority of the hybrid formulation.

Final remarks on the presented problems and future work projects are reported at the end of this thesis.

Chapter 1

General concepts: a brief overview

In this chapter we provide a brief overview of some basic concepts from integer and combinatorial optimization, graph theory, computational complexity, polyhedral theory and we briefly describe the Branch-and-Cut Algorithm. These concepts will turn out useful in the subsequent chapters of this dissertation.

1.1 Integer and Combinatorial Optimization

Integer and combinatorial optimization concerns the resolution of problems defined in a discrete space. The objective is to maximize or minimize a function of several variables, the so-called *objective function*, subject to inequality and equality constraints which with the integrality restrictions on some or all of the variables, define the *feasible solution set*.

Integer and combinatorial optimization is used to solve many problems of the everyday life. The problem of optimizing the productivity of a company, by managing the use of resources and by planning the production scheduling and the distribution of goods is only a first example of application. In mathematics there are applications in graph theory, statistic and logic. Lately, molecular biology has become a new interesting area of application.

In this thesis we consider only *linear integer programming problems*, namely we assume that the objective function and the constraints are linear and the variables are integer. In most cases, solving an integer programming problem is not easy.

Section 1.2 will be devoted to clarify what *easy* means and to find out whether a problem is *easy* or *hard*.

For more information about Integer and Combinatorial Optimization we refer to the books of Wolsey [55], and Nemhauser and Wolsey [42].

1.2 Computational complexity

In this section we summarize the most important concepts of complexity theory. To this end we refer to the books of Schrijver [48], Wolsey [55] and Nemhauser and Wolsey [42]. This theory deals with how much may be difficult a problem to solve. Informally a *problem* is a question, or a task, in fact there are two different types of problems: those that ask to find a solution and those that require only a 'yes' or 'no' answer. In the remainder of this section, we restrict ourselves to the last type of problems, and we refer to them as *decision problems*. This is not that much of a restriction, since the most problems can be reformulated as a decision problem. Usually a problem has several parameters defining it and by assigning them numerical values we obtain an *instance* of the problem. The *size of a problem instance* is the amount of information needed to represent it.

An *algorithm* is a list of instructions to solve every instance of a problem. The *running time* of an algorithm is the maximum number of steps that the algorithm needs to solve an instance of a given size s. It is estimated by an upper bound on the number of elementary operations such as additions, multiplications, comparisons, and so on, by assuming that each elementary operations is done in unit time. We say that a function $\gamma(n)$ is $O(\Gamma(n))$ when there exists a constant $c > 0$ and an integer value n_0 such that $|\gamma(n)| \leq c|\Gamma(n)|$ for all $n \geq n_0$. An algorithm is said to be *polynomial* if its time complexity function is $O(\pi(n))$, where $\pi(n)$ is polynomial, namely if the running time is polynomially bounded, otherwise it is said to be *exponential*. Decision problems for which a polynomial time algorithm exists are considered to be 'easy', their class is denoted by $\mathcal{P}$.

The class $\mathcal{NP}$ consists of all those problems such that for any input that has a positive answer, is possible to verify in polynomial time if such answer is correct. This definition does not imply that a problem belonging to $\mathcal{NP}$ is polynomially solvable, but just that the correctness of an answer can be verified in polynomial time. Obviously, $\mathcal{P} \subseteq \mathcal{NP}$, but $\mathcal{P} \neq \mathcal{NP}$ remains an open question.

The $\mathcal{NP}$-complete problems are defined to be the hardest problems in the class $\mathcal{NP}$. To well understand this statement we need to introduce the concept of *polynomial reduction.* A problem Φ_1 can be polynomially reduced to anther problem Φ_2 if there exists a polynomial time algorithm that transforms each instance ϕ_1 of Φ_1 into an instance ϕ_2 of Φ_2, namely for every instance of Φ_1 the answer is yes if and only if the answer for the corresponding instance of Φ_2 is yes.

Definition 1.1. *A decision problem Φ is called $\mathcal{NP}$-complete if it belongs to $\mathcal{NP}$ and any other problem in $\mathcal{NP}$ can be polynomially reduced to Φ.*

Note that, as previously observed, the majority of the problems can be reformulated as a decision problem, in particular, optimization problems (in which the aim is to maximize or minimize the objective function) can be reformulated as decision problems by fixing a bound on the value to be optimized. If the optimization problem is easy, the related decision problem belongs to $\mathcal{P}$. Conversely, if the decision problem belongs to the class $\mathcal{NP}$, then the corresponding optimization problem is hard to solve. Note that, while linear programming belongs to $\mathcal{P}$, integer linear programming is $\mathcal{NP}$-complete, i.e. it belongs to the most difficult problems in the class $\mathcal{NP}$.

1.3 Graph theory

In this section we provide some basic concepts from graph theory. To this end we refer to the paper of Gribkovskaia, Halskau and Laporte [25], *The bridges of Königsberg—a historical perspective*, and to the book of Bondy and Murty [2]. Many real-world situations can conveniently be described and modeled on graphs, i.e a set of points together with lines, joining some or all these points.

In 1973 the Swiss mathematician Leonhard Euler (1707-1783) solved the problem of The Seven Bridges of Königsberg, that is one of the most famous problems in graph theory, laying the foundation for modern graph theory. The city of Königsberg in Prussia (now called Kaliningrand, Russia), was crossed by the river Pregel, and included two islands connected to each other and to the mainland by seven bridges (Figure 1.1). In the 18^{th} century, the inhabitants of Königsberg

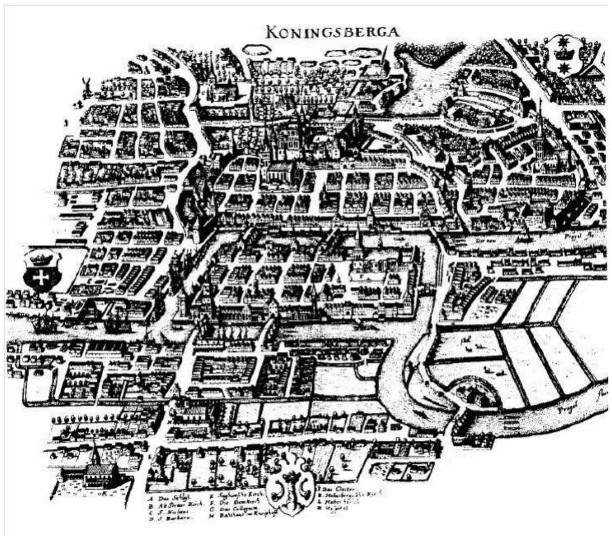


Figure 1.1: Map of Königsberg in Euler's time.

tried to discover whether there existed a closed walk that crossed exactly once each of the seven bridges. Euler represented each land mass with a point, called vertex or node, and each bridge with an line, called edge or arc (Figure 1.2) and proved that no such walk exists (see [17]). He proved that a walk can exists in a undirected graph such this, if the graph is connected and each vertex has even degree. Nowadays, a graph containing a closed walk crossing each edge exactly

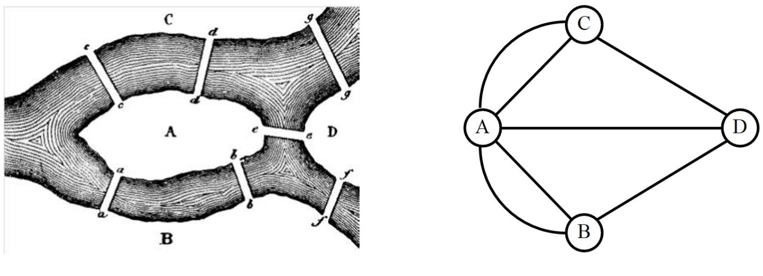


Figure 1.2: Euler's representation of the bridges of Königsberg.

once is called *Eulerian graph.*

An *undirected graph* $G = (V, E)$ consists of a finite set of *vertices* V and a finite set of *edges* E. An edge is a non-ordered pair of vertices. We will denote by n the number of vertices and by m the number of edges of the graph. A *complete graph*

is a graph with an edge between every pair of vertices.
If $G = (V, E)$ is a graph and $e = (v, u) \in E$, we say that v and u are *adjacent*, and that e is *incident* to v and u. The set of all edges incident to v is denoted by $\delta(v)$, and the *degree* of vertex v is the number of edges incident to it, that is $|\delta(v)|$. Note that, if a vertex has no incident edges, it is called *isolated* and its degree is zero. Given a set $S \subseteq V$, we denote by $E(S)$ the set of all edges with both end vertices in S, i.e. $E(S) = \{e = (v, u) \in E : v, u \in S\}$. Moreover, we denote by $\delta(S)$ the set of edges having one end vertex in S and the other in $V \setminus S$, that is, $\delta(S) = \{e = (v, u) \in E : v \in S, u \in V \setminus S\}$.
A subgraph of G is another graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. If $V' = V$, then G' is called *covering* (*spanning*) *subgraph*. If $E' = E(V')$, then G' is called *subgraph induced* by V'.
A *path* P from v to u in G is a sequence of distinct vertices $v_1, v_2, \ldots, v_k$ such that $v = v_1$, $u = v_k$ and $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, k-1$, namely $P = (V_P, E_P)$ is a subgraph of G such that $V_P = \{v_1 = v, v_2, \ldots, v_k = u\}$ and $E_P = \{(v_i, v_{i+1}) \in E : i = 1, \ldots, k-1\}$. The vertices v and u are called *end points* of the path P and k is the *length* of the path. A graph is connected if and only if there exists a path linking every pair of vertices of V. The maximal connected subgraphs of a graph are called *connected components*.
A *cycle* C of G is a closed path, that is a path whose end points coincide. A cycle is called *Hamiltonian* if it visits all the vertices of the graph, i.e. it is a covering of the graph.
A *tree* T is a connected graph that does not contain cycles. A vertex v is called a *leaf* of a tree T if it has degree equal to one in T. A forest F is an acyclic graph. Note that a connected forest is a tree. A *spanning forest* $F = (V_F, E_F)$ of $G = (V, E)$ is a forest with $V_F = V$. A spanning tree $T = (V_T, E_T)$ of $G = (V, E)$ is a tree having $V_T = V$ and $E_T \subset E$.
A *direct graph* is a pair $G = (V, A)$, where V is a set of vertices and A is a set of *arcs*, i.e. ordered pairs of vertices.
An *edge-colored graph* (labeled graph) is a graph $G = (V, E, L)$ such that to each edge is assigned a color from the finite set L of l colors. A *rainbow graph* is an edge-colored graph such that all edges have different color. In particular, if we denote by $\ell : E \to L$ the coloring function assigning to each edge a color from the

finite set L, the graph $G = (V, E, L)$ is rainbow if $|\ell(E)| = |E|$.

1.4 Polyhedral theory

Let $\mathbb{R}^k$ be the Euclidean linear space of dimension k, i.e. the set of k-dimensional vectors.

Definition 1.2. *Let $t > 0$ and $x_1, \dots, x_t \in \mathbb{R}^k$. A point $x \in \mathbb{R}^k$ is a linear combination of x_i, $i = 1, \dots, t$, if $x = \sum_{i=1}^{t} \lambda_i x_i$, for some $\lambda_1, \dots, \lambda_t \in \mathbb{R}$. Moreover, if $\sum_{i=1}^{t} \lambda_i = 1$, x is an affine combination. An affine combination is called convex combination if $\lambda_i \geq 0$, $i = 1, \dots, t$.*

In integer programming one of the main objectives is to find a linear inequality description of the set of feasible points.

Definition 1.3. *Given a set $S \subseteq \mathbb{R}^k$, the convex hull of S, denoted by $conv(S)$, is the set of all points that are convex combination of points in S.*

Given a set S of integral points, finding an inequality description of $conv(S)$ is not easy and knowing the dimension of $conv(S)$ as well as which inequalities are necessary for the description of $conv(S)$ are very important. Figure 1.3 shows a

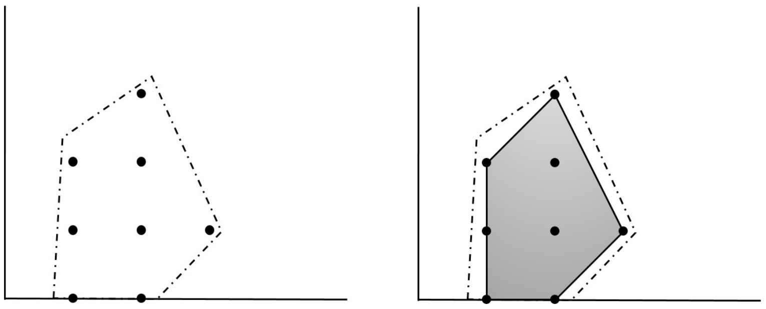


Figure 1.3: The black dots represent the set of integral points S. On the left a set containing S, on the right $conv(S)$.

generic hull containing a set of integral points in $\mathbb{R}^2$ on the left and the convex hull on the right.
In this section we will provide some basic results from linear algebra and some results concerning polyhedra.

Definition 1.4. *A set of points $x_1, \dots, x_t \in \mathbb{R}^k$ is linearly independent if the unique solution of $\sum_{i=1}^{t} \lambda_i x_i = 0$ is $\lambda_i = 0$, $i = 1, \dots, t$.*

Note that the maximum number of linearly independent points in $\mathbb{R}^k$ is k.

Definition 1.5. *A set of points $x_1, \dots, x_t \in \mathbb{R}^k$ is affinely independent if the unique solution of $\sum_{i=1}^{t} \alpha_i x_i = 0$, $\sum_{i=1}^{t} \alpha_i = 0$, is $\alpha_i = 0$, $i = 1, \dots, t$.*

Linear independent implies affine independent, but the converse is not true.

Proposition 1.1. *Let $x_1, \dots, x_t$ be t points in $\mathbb{R}^k$. Then the following statements are equivalent:*

- *The t points are affinely independent.*
- *The $t-1$ points $x_2 - x_1, \dots, x_t - x_1$ are linearly independent.*
- *The t points $(x_1, -1), \dots, (x_t, -1) \in \mathbb{R}^{k+1}$ are linearly independent.*

It is easy to see that the maximum number of affinely independent points in $\mathbb{R}^k$ is $k+1$, i.e. k linear independent points and the zero vector.

Definition 1.6. *A polyhedron $S \subseteq \mathbb{R}^k$ is a set of points that satisfy a finite set of linear inequalities, namely $S = \{x \in \mathbb{R}^k : Ax \leq b\}$, where $A \in \mathbb{R}^{h \times k}$ and $b \in \mathbb{R}^h$. A bounded polyhedron is called polytope.*

Note that some inequalities defining S can be equations.

Definition 1.7. *A polyhedron $S = \{x \in \mathbb{R}^k : Ax \leq b\}$ is full-dimensional if $dim(S) = k$.*

Let $M = \{1, \dots, m\}$, $M^{=} = \{i \in M : a^i x = b_i \text{ for all } x \in S\}$ and $M^{\leq} = \{i \in M : a^i x < b_i \text{ for some } x \in S\} = M \setminus M^{=}$. Let $(A^{=}, b^{=})$ and $(A^{\leq}, b^{\leq})$ the corresponding rows of (A, b). According to this notation, the following proposition holds true:

Proposition 1.2. *If $S \subseteq \mathbb{R}^k$, then $dim(S) + rank(A^{=}, b^{=}) = k$.*

Given a polyhedron $S = \{x \in \mathbb{R}^k : Ax \leq b\}$, there may exist inequalities $a^i x \leq b_i$ that are not necessary for the description of S and can therefore be dropped.

Definition 1.8. *Given an inequality $\pi x \leq \pi_0$, we say that $\pi x \leq \pi_0$ is a valid inequality for S if it is satisfy by all points in S.*

It is easy to see that if $\pi x \leq \pi_0$ is a valid inequality, $S \subseteq \{x \in \mathbb{R}^k : \pi x \leq \pi_0\}$.

Definition 1.9. *If $\pi x \leq \pi_0$ is a valid inequality for S and if $F = \{x \in S : \pi x = \pi_0\}$, F is called a face of S. Moreover, if $F \neq \emptyset$ and $F \neq S$, then F is called a proper face induced by $\pi x \leq \pi_0$. A facet is a maximal proper face, namely is a proper face F such that $dim(F) = dim(S) - 1$. A vertex of S is a face having dimension zero.*

Definition 1.10. *A linear system that define a polyhedron S is called minimal when:*

i. by setting an inequality to equality the polyhedron is reduced, and

ii. by removing an inequality or an equality the polyhedron changes.

Example 1.1. *Suppose $S \subset \mathbb{R}^2$ is given by*

$$\begin{array}{rcrcl}
x_1 & & & \leq & 3 \\
2x_1 & + & 3x_2 & \leq & 15 \\
x_1 & - & x_2 & \leq & 3 \\
 & & x_2 & \leq & 5 \\
3x_1 & + & 2x_2 & \geq & 6 \\
x_1 & & & \geq & 0 \\
 & & x_2 & \geq & 0
\end{array}$$

see Figure 1.4.
S is full-dimensional since $(2,0)$, $(0,3)$, $(3,3)$ lie in S and are affinely independent. Note that $(0,3)$ and $(3,3)$ are two affinely independent points that satisfy the inequality $x_1 \leq 3$ as equality, therefore $x_1 \leq 3$ is a facet. Similarly the inequalities $2x_1 + 3x_2 \leq 15$, $3x_1 + 2x_2 \geq 6$, $x_1 \geq 0$ and $x_2 \geq 0$ are facets for S. Moreover $x_2 \leq 5$ is a face containing only one point in S, hence, it is redundant. Inequality $x_1 - x_2 \leq 3$ can be obtained as the sum of $x_1 \leq 3$ and $-x_2 \leq 0$, therefore it is another example of redundant inequality.

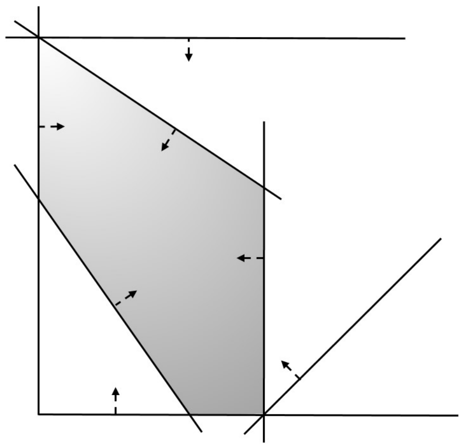


Figure 1.4: Linear system defining $S \subset \mathbb{R}^2$.

The minimal description for S is given by

$$\begin{array}{rcrcl} x_1 & & & \leq & 3 \\ 2x_1 & + & 3x_2 & \leq & 15 \\ 3x_1 & + & 2x_2 & \geq & 6 \\ x_1 & & & \geq & 0 \\ & & x_2 & \geq & 0. \end{array}$$

The following theorem is useful to establish whether a valid inequality is a facet.

Theorem 1.1. *Let $(A^=, b^=)$ be the equality set of $S \subseteq \mathbb{R}^k$ and let $F = \{x \in S : \pi x = \pi_0\}$ be a proper face of S. The following two statements are equivalent:*

- *F is a facet of S.*
- *If $\lambda x = \lambda_0$ for all $x \in F$ then*

$$(\lambda, \lambda_0) = (\alpha\pi + uA^=, \alpha\pi_0 + ub^=) \text{ for some } \alpha \in \mathbb{R} \text{ and some } u \in \mathbb{R}^{|M^=|}.$$

For more details on polyhedral theory we refer to the textbooks of Schrijver [48], Wolsey [55] and Nemhauser and Wolsey [42].

1.5 Branch-and-Cut Algorithm

In this section we introduce and describe the *branch-and-cut algorithm*, that is an algorithm for solving integer linear programming problems. The branch-and-cut algorithm, introduced by Padberg and Rinaldi [43] consists of a combination of the *branch-and-bound algorithm* and the *cutting plane method.*
The branch-and-bound algorithm was designed by Dakin [14] by modifying an earlier method of Land and Doig [31]. This method works by solving a sequence of linear programming relaxations of the integer programming problem by following a *divide-et-impera* approach. The idea is to solve the integer linear programming problem by solving smaller and easier subproblems, obtained recursively partitioning (branching) the feasible solution set. The branch-and-bound approach differs from a complete enumeration method as it tries to explore only promising areas of the feasible solution set by using the upper and the lower bounds on the optimal value of the solution to exclude a priori certain subproblems. In consequence, having good lower and upper bounds for the optimal value is essential for the effectiveness of the algorithm. This procedure can be graphically represent by a tree (the branch-and-bound tree) such that the root node represents the initial integer linear programming and each node of the tree is a particular subproblem. The leaves of the tree represent the active nodes, i.e. the subproblems that still need to be solved.
The cutting plane method has been developed by Gomory [24] to solve linear integer programming. Given an integer linear program (ILP), the first step is to identify an optimal solution of the linear programming relaxation (RL) and check whether this solution is integer or not. If the solution is integer, it is an optimal solution for ILP, otherwise, the idea is to find and add one or more valid inequalities (cuts), which are satisfied by all integer feasible solutions of ILP, but not by the optimal solution of RL, and solve the linear programming relaxation of the problem obtained by adding the cuts. The method will keep appending cuts and solving the new linear programming relaxation until the solution will not be integer or it will conclude that the problem is not feasible. If an integer solution exists, the cutting plane method will find it by adding a finite number of cuts, the problem is that such number could be very high.

1. **Initialization** Denote the initial integer programming problem by P^0 and set the active nodes to be $L = P^0$. Set the upper bound to be $ub = \infty$ and $\underline{z_l} = -\infty$ for the one problem $l \in L$.

2. **Termination** If $L = \emptyset$, then the solution x^* associated with the incumbent objective value ub is optimal. Note that if no such x^* exists, i.e. if $ub = \infty$, then the integer linear programming is infeasible.

3. **Subproblem selection** Select and delete a subproblem P^l from L.

4. **Relaxation** Solve the linear programming relaxation of P^l.
    - If the relaxation is infeasible, set $\underline{z_l} = \infty$ and go to Step 6.
    - If the relaxation is feasible and the optimal objective value is finite, set $\underline{z_l}$ equal to this value and let x^l_{RL} be an optimal solution; otherwise set $\underline{z_l} = -\infty$.

5. **Add cutting planes** Search for cutting planes that are violated by x^l_{RL}; if one or more cuts are found, add them to the linear programming relaxation of P^l and go to Step 4.

6. **Updating and Pruning**
    - If $\underline{z_l} \geq ub$, go to Step 2.
    - If $\underline{z_l} < ub$, and x^l_{RL} is integer, update $ub = \underline{z_l}$, delete from L all subproblems $\bar{l}$ with $\underline{z_{\bar{l}}} \geq ub$ and go to Step 2.

7. **Partitioning** Let S^l_j, $j = 1, \ldots, k$, be a partition of the constraint set S^l of the subproblem P^l and let P^l_j, $j = 1, \ldots, k$, be P^l with feasible region restricted to S^l_j. Add $\{P^l_1, \ldots, P^l_k\}$ to L and set $\underline{z_{l_j}}$, $j = 1, \ldots, k$, to the value of $\underline{z_l}$ for the parent problem l. Go to Step 2.

Figure 1.5: Branch-and-Cut Algorithm.

The branch-and-cut algorithm arises from an attempt to overcome the limitations of both the branch-and-bound algorithm and the cutting plane method. At each vertex, i.e. at each subproblem, of the branch-and-bound tree new cuts are generated and added to find an integer feasible solution or at least to improve the bound. When the new cuts turn out to be ineffective a new branching is performed. The step-by-step description of the branch-and-cut algorithm for a minimization problem is summarized in Figure 1.5.

For more details on branch-and-bound, cutting planes and branch-and-cut algorithms we refer to the paper of Mitchell [41], *Branch-and-Cut Algorithms for Combinatorial Optimization Problem*, and the book of Wolsey [55].

Chapter 2

Spanning Tree and Cycle Cover

2.1 Introduction

The aim of this chapter is to present some Spanning Tree and Cycle Cover problems. Some particular versions of these problems are standard topics in the combinatorial optimization literature and arise frequently as subproblems in more general problems. The *Minimum Spanning Tree Problem* and the *Traveling Salesman Problem* are an example of Spanning Tree and Cycle Cover, respectively. These problems are among the best known and intensively studied combinatorial optimization problems.

2.2 Spanning Trees

Given a connected undirected graph $G = (V, E)$, with $n = |V|$ vertices and $m = |E|$ edges, and given a function $w : E \to \mathbb{R}^+$, assigning to each edge $e \in E$ a weight $w_e \in \mathbb{R}^+$, the *Minimum Spanning Tree Problem* (MSTP) aims to find a spanning tree T of G having the minimum total weight (see Figure 2.1). The total weight of a tree is the sum of the weights of the edges of the tree. The MSTP has several applications in the design of networks, including computer, telecommunication, transportation networks, etc. Moreover it occurs as a subproblem in the solution of others problem, and its algorithms are used in several exact and heuristics algorithms for solving the traveling salesman problem, the matching problem, and so on.

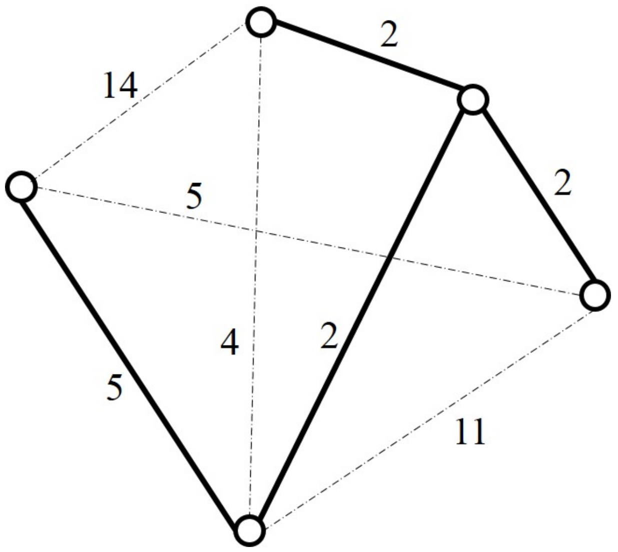


Figure 2.1: The Minimum Spanning Tree: an example.

Given the connected undirected grapg $G = (V, E)$, the MSTP can be formulated as a linear program (LP) [16] with variables x_e, $e \in E$, as follows

$$\text{minimize } z = \sum_{e \in E} w_e x_e \tag{2.1}$$

subject to

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad S \subset V,\ |S| \geq 2 \tag{2.2}$$

$$\sum_{e \in E} x_e = n - 1 \tag{2.3}$$

$$x_e \geq 0 \qquad e \in E, \tag{2.4}$$

where $E(S)$ denotes all edges of E which have both ends in the vertex set S. In this formulation, constraints (2.2) are subtour elimination constraints [15], they guarantee that the optimal solution contains no cycles. Constraint (2.3) imposes that the solution contains exactly $n - 1$ edges. Constraints (2.4) ensure that the variables can assume only positive values. In spite of the fact that the linear program has an exponential number of constraints, the LP can be solved in polynomial time by the greedy algorithm of Kruskal [30] or the greedy algorithm of Prim [45]. Moreover, Edmonds [16] proved that the extreme points of the constraint set of LP correspond exactly to the incidence vector of trees in G and that, if we assume that the graph G is complete, all the constraints define facets of the associated polytope.

Many real world problems can be modeled as Spanning Trees with the addition of some new requests. For this reason the literature contains many constrained versions of the STP.

The *Steiner Tree Problem* (STP) can be considered a generalized MST. Given a connected graph G, suppose that the set of vertices is partitioned into two set, the set of *terminal* vertices and the set of *Steiner* vertices. The STP consists of determining a minimum cost tree spanning all terminal vertices and, if necessary, some Steiner vertices. The Steiner vertices do not need to be spanned, but they may belong to the optimal solution if their inclusion reduces the solution cost (see, for example, [23], [26]). The problem was first studied for Euclidean distance metric [26]. Given a set N of n points in the Euclidean plane, the shortest network interconnecting N is called a Euclidean Steiner Minimal Tree (ESMT) for N. The MSTP is a special case of the STP, where all vertices are terminal. Unlike the MSTP, that can be solved in polynomial time, the STP is $\mathcal{NP}$-hard [20].

Given a connected undirected graph G, the *Maximum Leaf Spanning Tree Problem* (MLSTP) [19], [35] is to find a spanning tree whose number of leaves, i.e. the vertices having degree equal to 1 in the tree, is maximum. The problem, which has applications in communication networks and circuit layouts, is trivial to solve for complete graphs. However, for the general case, when the graph is sparse, it known to be $\mathcal{NP}$-hard [21].

Given a connected undirected graph G, the *Degree Preserving Spanning Tree Problem* (DPSTP) [13] asks for a spanning tree of G with the maximum number of vertices having the same degree in the solution tree and in the graph G. These vertices are called *full degree vertices* and the problem is also known in the literature as the *Full Degree Spanning Tree Problem* [1]. The main real application of the DPSTP originates from the *Vertex Feedback Edge Set Problem* (VFESP), which can be seen as the complementary problem of the DPSTP. The VFESP is to find a complement of a tree (cotree) of G that is incident on the minimum number of vertices. It is easy to see that the vertices that are not incident to the edges of a co-tree have full degree in the tree.

2.2.1 Spanning Tree on edge-labeled graphs

In this subsection we describe two known combinatorial problems defined on edge-colored graphs.

Given a connected undirected edge-colored graph $G = (V, E, L)$, the *Minimum Labeling Spanning Tree Problem* (MLSTP) [11] asks for a spanning tree of G having as few colors as possible. The colors (labels) on the edges can represent different transportation modes in a transportation networks, or connections belonging to different competing companies in the case of a telecommunication network. In [11] the authors proved that the MLSTP is $\mathcal{NP}$-hard by reduction from the Set Covering Problem. Independently, in [4] the authors proved that the MLSTP is $\mathcal{NP}$-hard by reduction from the Dominating Set Problem.

Given a positive integer k and a connected undirected graph $G = (V, E, L)$, whose edges are colored, the *k-Labeled Spanning Forest Problem* (KLSFP) [9] consists of finding the minimum number of connected components that can be obtained when there is a constraint on the maximum number k of colors that can be used. The problem, which finds application in telecommunication network, generalizes the MLSTP. It is easy to see that a spanning tree would be the optimum if it can be found by using at most k colors. In [9] the authors showed that the KLSFP is $\mathcal{NP}$-complete by observing that the MLST is a special case of it.

2.3 Cycle Cover Overview

Given a connected directed graph $D = (V, A)$, with $n = |V|$ vertices and $m' = |A|$ edges, and given a function $c : A \to \mathbb{R}^+$, assigning to each arc $a = (v, u) \in A$ a cost (or distance) $c_{vu} \in \mathbb{R}^+$, the *Traveling Salesman Problem* (TSP) consists of finding a minimum distance cycle passing through each vertex exactly once, i.e. the shortest Hamiltonian cycle in the graph (see Figure 2.2). If $c_{vu} = c_{uv}$, the problem is called

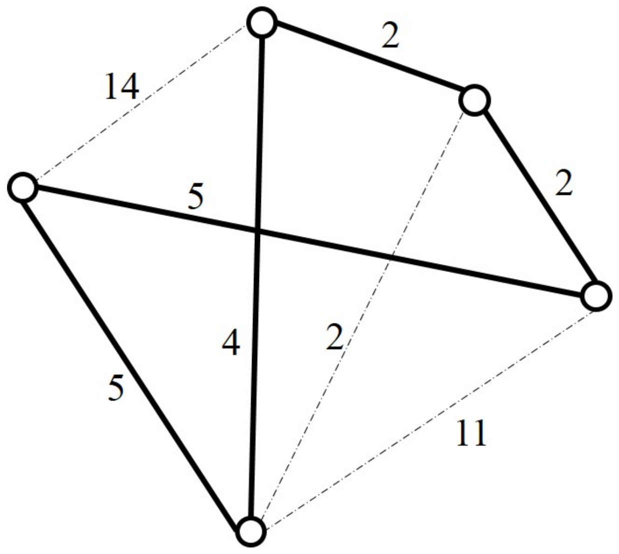


Figure 2.2: The Traveling Salesman Problem: an example.

symmetric traveling salesman problem, otherwise it is called *asymmetric*. The most known application of the TSP is that of a traveling salesman that wants to visit a set of towns exactly once, starting from and returning to his home town, and wants to do it using the shortest possible tour. This problem underlies several vehicle routing applications and there are many practical problems that can be formulated as TSP problems.

The TSP has been one of the first problems to be proven $\mathcal{NP}$-complete by reduction from the Hamiltonian Cycle Problem. One of the first integer linear formulations for the TSP was proposed by Dantzing, Fulkerson and Johnson [15]. It associates one binary variable z_a to every arc $a \in A$, equal to 1 if the arc belongs to the optimal solution. The formulation is

$$\text{minimize } z = \sum_{a \in A} c_a z_a \tag{2.5}$$

subject to

$$\sum_{a \in \delta^-(u)} z_a = 1 \qquad u \in V \tag{2.6}$$

$$\sum_{a \in \delta^+(v)} z_a = 1 \qquad v \in V \tag{2.7}$$

$$\sum_{a \in A(S)} z_a \leq |S| - 1 \qquad S \subset V,\ 2 \leq |S| \leq n-2 \tag{2.8}$$

$$z_a \in \{0, 1\} \qquad a \in A \tag{2.9}$$

where $\delta^+(w) = \{(v,u) \in A : v = w\}$, $\delta^-(w) = \{(v,u) \in A : u = w\}$ and $A(S)$ denotes all arcs of A having both ends in the vertex set S. Constraints (2.6) and (2.7) are degree constraints, they impose that each vertex is visited exactly once and is left exactly once, respectively. Constraints (2.8) ensure the formation of a tour only on the set V. Because of degree constraints, subtours over one vertex and over $n-1$ vertices cannot occur. Therefore constraints (2.8) are defined for $2 \leq |S| \leq n-2$ only. Finally, constraints (2.9) impose binary conditions on the z variables. For more details we refer to the paper *The Traveling Salesman Problem* of Jünger, Reinelt and Rinaldi [28] and the paper *The Traveling Salesman Problem: An overview of exact and approximate algorithms* of Laporte [32].

The *Vehicle Routing Problem* (VRP) can be seen as a natural generalization of the TSP. Given a connected directed graph D, suppose that the set V of n vertices represents $n-1$ *cities* $(v_2, \ldots, v_n)$, and a *depot* (v_1) and suppose that to each arc (v,u) is associated a non-negative distance cost c_{vu}. Moreover, let k be the number of available vehicles based ot the depot. The value k can be fixed a priori or it can be a decision variable. If k is a decision variable, it makes sense to associate a fixed cost on the use of a vehicle. The VRP [33] is the problem of identify optimal delivery routes from the depot v_1 to the cities, in such a way that each city is visited exactly once by only one vehicle and all vehicles have to end at the depot. Note that there could be some side constraints that have to be satisfied, for example:

- *capacity restrictions*: to each city v_i can be associated a demand d_i, therefore it is necessary to associate to each vehicle a set of vertices such that the sum of the corresponding demands does not exceed the vehicle capacity;
- *route restrictions*: in a route a vehicle cannot visit more than a fixed number q of cities;
- *time windows*: some city must be visited within a fixed time interval;
- *precedence relations*: may be required that a city must be visited before another one.

It is easy to see that if $k = 1$ and all the side constraints are relaxed, we obtain the TSP, therefore the VRP is another example of $\mathcal{NP}$-complete cycle cover problem.

Given a connected undirected graph $G = (V, E)$, non-negative edge costs c_e, for all $e \in E$, and a positive integer p the *Hamiltonian p-Median Problem*, introduced by Branco and Coelho [3], consists of finding p disjoint cycles of minimum total cost, covering all vertices of the graph G. The problem, that finds applications in the context of depot location or multi-depot vehicle routing, coincides with the TSP when $p = 1$. For this relation, it follows that the HpMP is $\mathcal{NP}$-hard.

The *Steiner Cycle Problem* (STP) is closely related to the Traveling Salesman Problem and the Steiner Tree Problem. Given a connected undirected graph G, suppose that the set of vertices V is partitioned into two subsets, the set of *required* vertices and the set of *optional* vertices. Assume that to each edge is associated a cost and to each optional vertex a non-negative penalty. The STP [46] asks for a simple cycle in G visiting at least the required vertices and such that the sum of the costs of the edges in the cycle plus the penalties of the optional vertices that does not belong to the cycle is minimum. The problem has applications in the optimal design of telecommunication systems and transportation networks, and in [46] the author showed that it is strongly $\mathcal{NP}$-hard since the TSP is a particular case.

2.3.1 Cycle Cover on edge-labeled graphs

In this section we describe a cycle cover problem defined on edge-colored graph and two variants.

Given a connected undirected edge-colored graph $G = (V, E, L)$, the *Minimum Labeling Hamiltonian Cycle Problem* (MLHCP) [8], also known in the literature as the *Colorful Traveling Salesman Problem* (CTSP), asks for a Hamiltonian cycle with the least number of labels (colors). Colors can represent transporters, types of telecommunication fibers or production technologies [8], [56]. The *Minimum Labeling Hamiltonian Cycle Problem with Length Constraint* (MLHCPLC) and the *Label Constrained Traveling Salesman Problem* (LCTSP), are two variants of the MLHCP [27] defined on a connected undirected edge-colored graph with edge costs. In the MLHCPLC the tour cannot be longer than a given length, instead in the LCHCP the objective is to minimize the tour length with a constraint on the maximum number of colors that can be used.

Chapter 3

The Rainbow Cycle Cover Problem

“ *Whenever the rainbow appears in the clouds, I will see it and remember the everlasting covenant between God and all living creatures.* ”

Genesis 9:16

3.1 Introduction and Problem Description

In this chapter we address *The Rainbow Cycle Cover Problem* (RCCP), that represents another example of cycle cover problem, defined on a connected undirected edge-colored graph. A slightly different version of this chapter was submitted to Networks. The main contribution of the chapter is to present a mathematical model and a branch-and-cut algorithm for the RCCP.
Let $G = (V, E, L)$ be a connected, undirected and edge-colored graph, where V is the set of n vertices, E is the set of m edges, and L is a set of l colors. Let $\ell : E \to L$ be a coloring function assigning to each edge a color from the set L. A *rainbow cycle* of G is a cycle $C = (V_C, E_C)$, where $V_C \subseteq V$ and $E_C \subseteq E$, in which all edges have different colors, i.e. $|\ell(E_C)| = |E_C|$. A *rainbow cycle cover* (*RCC*) of G is a collection of rainbow cycles such that each vertex of the graph G belongs to exactly one cycle. Note that in this context a single vertex is considered

as a degenerate rainbow cycle. From now on, we will refer to these as the *trivial* rainbow cycles. The RCCP asks for a *RCC* with the least number of rainbow cycles. Li and Zhang [34] investigated the complexity of the rainbow tree, cycle and path partition problems and proved that identifying a *RCC* with the minimum number of cycles is $\mathcal{NP}$-hard. To the best of our knowledge, no mathematical formulation for this problem has ever been put forward. In this work we propose an integer mathematical formulation and valid inequalities that will be used within a branch-and-cut algorithm. We will also consider some properties that a rainbow cycle cover must satisfy. These properties will allow us to preprocess the instances and add some ad hoc constraints that will help to solve the problems, sometimes in a very effective way. The remainder of this chapter is organized as follows. The mathematical formulation and the description of the properties that a *RCC* must satisfy are presented in Section 3.2. Section 3.3 contains the valid inequalities. The branch-and-cut algorithm is described in Section 3.4. Section 3.5 contains the computational results.

3.2 Mathematical formulation and Properties

In this section we present an integer linear mathematical formulation for the RCCP. Since the objective is to minimize the number of rainbow cycles needed to cover the vertices of the graph, we introduce the set of binary variables γ_c, for $c = 1, \ldots, \bar{c}$ associated with each non-trivial cycle c of a *RCC*, whose value is equal to 1 if and only if c contains at least three vertices. It is easy to see that the number of variables γ_c depends on the number of possible non-trivial cycles, so it is useful to compute a good upper bound $\bar{c}$ on this number. Note that, according to the definition, the variables γ_c are equal to one if and only if c contains at least three vertices, therefore $\lfloor n/3 \rfloor$ is an obvious upper bound on the maximum number of non-trivial rainbow cycles that a *RCC* of a graph can contain. Moreover, we define binary variables y_v^c equal to 1 if and only if vertex v belongs to cycle c, and binary variables x_e^c equal to 1 if and only if edge e belongs to cycle c. In order to introduce constraints that can help prevent equivalent solutions, it is useful to define an index set $I_q = \{1, \ldots, q\}$, for an integer q, and to define the undirected graph

$G = (V, E, L)$, with vertex set $V = I_n$. The formulation is then as follows:

$$\text{minimize } z = \sum_{c=1}^{\bar{c}} \gamma_c + \sum_{v \in V} M\Big(1 - \sum_{c=1}^{\bar{c}} y_v^c\Big) \tag{3.1}$$

subject to

$$\sum_{v \in V} y_v^c \leq l\, \gamma_c \qquad c = 1, \dots, \bar{c} \tag{3.2}$$

$$3\,\gamma_c \leq \sum_{v \in V} y_v^c \qquad c = 1, \dots, \bar{c} \tag{3.3}$$

$$\sum_{c=1}^{\bar{c}} y_v^c \leq 1 \qquad v \in V \tag{3.4}$$

$$\sum_{e \in \delta(v)} x_e^c = 2\, y_v^c \qquad v \in V,\ c = 1, \dots, \bar{c} \tag{3.5}$$

$$x_e^c \leq y_v^c \qquad v \in V,\ e \in \delta(v) \tag{3.6}$$

$$\sum_{e \in \delta(S)} x_e^c \geq 2(y_v^c + y_u^c - 1) \qquad S \subset V,\ v \in S,\ \{v, u\} \in \{S, \bar{S}\},\ c = 1, \dots, \bar{c} \tag{3.7}$$

$$\sum_{e \in E_k} x_e^c \leq 1 \qquad c = 1, \dots, \bar{c},\ k \in L \tag{3.8}$$

$$\gamma_{c+1} \leq \gamma_c \qquad c = 1, \dots, \bar{c} - 1 \tag{3.9}$$

$$\sum_{c=v+1}^{\bar{c}} y_v^c = 0 \qquad v \in V : v < \bar{c} \tag{3.10}$$

$$y_v^c \leq \sum_{w < v} y_w^{c-1} \qquad v \in V \setminus \{1\},\ c = 3, \dots, \bar{c} \tag{3.11}$$

$$\gamma_c \in \{0, 1\} \qquad c = 1, \dots, \bar{c} \tag{3.12}$$

$$y_v^c \in \{0, 1\} \qquad v \in V,\ c = 1, \dots, \bar{c} \tag{3.13}$$

$$x_e^c \in \{0, 1\} \qquad e \in E,\ c = 1, \dots, \bar{c}, \tag{3.14}$$

where $\delta(v)$ denotes the set of edges incident to v in G, $\bar{S} = V \setminus S$, $\delta(S) = \{e = (v, u) \in E : v \in S\ u \in \bar{S}\}$, $\{S, \bar{S}\} = \{\{v, u\} : v \in S, u \in \bar{S}\}$ and $E_k = \{e \in E : \ell(e) = k\}$. Note that the set $\delta(S)$ contains only edges of the graph, whereas the set $\{S, \bar{S}\}$ contains all the possible pairs between S and $V \setminus S$.

Constraints (3.2) and (3.3) are logical constraints linking the binary variables γ_c with the binary variables y_v^c. Note that at most l vertices can belong to the same cycle since this is the number of different colors of the graph. Constraints (3.4) and (3.5) ensure that each vertex belongs to at most one cycle and, if it belongs to a cycle, it has a degree equal to 2 in that cycle. It is easy to see that to ensure the validity of the constraints (3.5), if a variable γ_c is equal to one, then there must be at least three vertices in the corresponding rainbow cycle. Constraints (3.6) impose that if a vertex is not in the cycle c, the edge incident on such vertex cannot belong to that cycle. Constraints (3.7) guarantee solutions with not more than one cycle associated to each variable γ_c. Constraints (3.8) ensure the rainbow property. These constraints impose that a cycle cannot contain two edges having the same color. Even if constraints (3.9), (3.10) and (3.11) are not necessary for the model, we insert them in the model because they help eliminate symmetries. Constraints (3.9), mean that there will never be a variable γ_{c+1} equal to one if γ_c is equal to zero, for any c. Constraints (3.10) impose that vertices with and index $v < \bar{c}$ cannot belong to a cycle c such that $c > v$. Moreover, constraints (3.11) mean that a vertex v can belong to a cycle of index c if and only if at least one vertex w with a lower index belongs to the cycle of index $c - 1$. The constraints (3.10) and (3.11) are a generalization of the symmetry constraints introduced by Fischetti et al. [18] in the context of the Vehicle Routing Problem. The objective function (3.1) requires the minimization of the number of rainbow cycles. To this end, we need an objective function with two terms. The first part minimizes the number of non-trivial cycles by minimizing the sum of the variables γ_c. The second part forces the vertices to belong to a cycle, whenever possible, giving a weight M to each isolated vertex. If we do not force the vertices to belong to a cycle, the optimal solution would be n trivial cycles and the optimal value would be zero.

3.2.1 Properties of a Rainbow Cycle Cover

In this subsection we present some properties that a *RCC* must satisfy. Let ζ_v denote the colored degree of vertex v, i.e. the number of different colors incidents to v. It is easy to see that if a vertex v has colored degree equal to one, i.e. $\zeta_v = 1$, then vertex v will be a trivial rainbow cycle, that is, it will be isolated:

$$\sum_{c=1}^{\bar{c}} y_v^c = 0 \qquad v \in V : \ \zeta_v = 1. \tag{3.15}$$

Therefore, all edges incident to that vertex will be equal to 0 in the optimal solution. In view of these observations, if we denote by n_1 the number of vertices whose colored degree is equal to one, then the upper bound on the number of variables γ reduces to $\lfloor (n - n_1)/3 \rfloor$.
This observation is easily extensible to the edges. Suppose that given an edge $e = (v, u)$, the total number of colors incident to v and u is equal to two, i.e. $|\{\ell(\delta(v)) \cup \ell(\delta(u))\}| = 2$, then edge e cannot belong to a rainbow cycle. Note that, if $|\{\ell(\delta(v)) \cup \ell(\delta(u))\}| = 2$, then one of these two colors must be the color of edge e, therefore there will be only one more color available to connect e with others edges of a *RCC* which is not possible. We can state this property through the constraints

$$\sum_{c=1}^{\bar{c}} x_e^c = 0 \qquad e = (v, u) \in E : \ |\{\ell(\delta(v)) \cup \ell(\delta(u))\}| = 2. \tag{3.16}$$

This two observations can be used in a preprocessing phase to reduce the size and the difficulty of the instances. Note that, the last property is included in the following most general one. Pairs of vertices having a colored degree equal to 2 and the same set of incident colors cannot belong to the same cycle:

$$y_v^c + y_u^c \leq 1 \qquad v, u \in V : \ell(\delta(v)) = \ell(\delta(u)), \ \zeta_v = \zeta_u = 2, \ c = 1, \ldots, \bar{c}. \tag{3.17}$$

Obviously, if two of these vertices are adjacent, the edge linking them cannot belong to a *RCC*, which means that all variables associated to the edge will be equal to zero and then constraints (3.16) are satisfied. Moreover, for each vertex v such that $\zeta_v = 2$, the following constraints are valid for the RCCP:

$$\sum_{e \in \delta_k(v)} x_e^c - \sum_{e \in \delta_h(v)} x_e^c = 0 \quad v \in V : \ \zeta_v = 2, \ \ell(\delta(v)) = \{k, h\}, \ c = 1, \ldots, \bar{c}, \tag{3.18}$$

where $\delta_k(v) = \{e \in E : e \in \{\delta(v) \cap E_k\}\}$, i.e. the set of all the edges incident to v and having color k.
Suppose now that the shortest cycle including two fixed vertices v and u contains at least $l+1$ edges, then it is easy to observe that the two vertices cannot belong to the same rainbow cycle, since a rainbow cycle cannot contain more than l edges. Identifying shortest cycles containing two fixed vertices can be efficiently achieved by means of Suurballe's algorithm ([53], [54]). This algorithm identifies two disjoint paths in a non-negative weighted directed graph so that both paths connect the same pair of vertices and have a minimum total length. Suurballe's algorithm uses Dijkstra's algorithm to find the first path. It then modifies the weights of the edges preserving their non-negativity. After this modification Suurballe's algorithm uses Dijkstra's algorithm a second time. Procedure 1 shows the details of Suurballe's algorithm.
A key point of the algorithm is line 3. The weights of the arcs are modified according

Procedure 1: Suurballe's algorithm

```
  Input: $G(V, A)$, $v \in V$, $u \in V$, $w(A)$
  Output: $SC(v, u)$ the shortest cycle containing $v, u$
1 $T_1 \leftarrow$ DijkstraAlgorithm$(v, G, w(A))$
2 $P_1 \leftarrow$ IdentifyPath$(v, u, T_1)$
3 $w(A) \leftarrow$ updateWeightEdge$(T_1, G)$
4 $G_{P_1} \leftarrow$ CreateResidualGraph$(P_1, G)$
5 $T_2 \leftarrow$ DijkstraAlgorithm$(v, G_{P_1}, w(A))$
6 $P_2 \leftarrow$ IdentifyPath$(v, u, T_2)$
7 $SC(v, u) \leftarrow$ mergePaths$(P_1, P_2)$
8 return $SC(v, u)$
```

to the following formula

$$w(i, j) = w(i, j) - d(v, j) + d(v, i) \qquad (i, j) \in A, \tag{3.19}$$

where v is the root vertex of the shortest path tree T_1, and $d(v, j)$, $d(v, i)$ are the distances from v to j and i in T_1. Thanks to Suurballe's algorithm we can impose the constraints

$$y_v^c + y_u^c \leq 1 \qquad v, u \in V : |SC(v, u)| \geq l + 1,\ c = 1, \ldots, \bar{c}, \tag{3.20}$$

where $SC(v, u)$ represents the shortest cycle containing the vertices v and u. Note that when two vertices v and u are incompatible, that is, when two vertices cannot belong to the same cycle because of (3.17) and (3.20), then the following inequalities are valid for the RCCP:

$$\sum_{e \in \{\delta(v) \cup \delta(u)\}} x_e^c \leq 2 \qquad v, u \in V : y_v^c + y_u^c \leq 1,\ c = 1, \ldots, \bar{c} \tag{3.21}$$

$$\sum_{e \in \{\delta_k(v) \cup \delta_k(u)\}} x_e^c \leq 1 \qquad v, u \in V : y_v^c + y_u^c \leq 1,\ k \in L,\ c = 1, \ldots, \bar{c}. \tag{3.22}$$

A last observation about the properties of a *RCC* is the following: given a color k, if the edges of color k are incident to f_k distinct vertices, i.e. if $|\{v \in V : \delta(v) \cap E_k \neq \emptyset\}| = f_k$, then only $\lfloor f_k/2 \rfloor$ edges of color k can belong to a *RCC*. It is sufficient to select one more edge and there will be at least two edges having the same color and incident to a same vertex. We can therefore impose the following constraints:

$$\sum_{c=1}^{\bar{c}} \sum_{e \in E_k} x_e^c \leq \lfloor f_k/2 \rfloor \qquad k \in L. \tag{3.23}$$

3.3 Valid inequalities

In the previous section we proposed a mathematical formulation for the Rainbow Cycle Cover Problem and we also provided some simple properties that a *RCC* has to satisfy. We now present some valid inequalities for the RCCP.

Proposition 3.1. *The constraints*

$$y_v^c - \gamma_c \leq 0 \qquad v \in V,\ c = 1, \ldots, \bar{c} \tag{3.24}$$

$$x_e^c - \gamma_c \leq 0 \qquad e \in E,\ c = 1, \ldots, \bar{c} \tag{3.25}$$

are satisfied by all optimal RCCP solutions.

Proof. These constraints state that the variable representing a cycle has to be equal to 1 if a vertex or an edge belongs to that cycle. □

An example of a solution violating constraints (3.24) but satisfying constraints (3.2) is the following: $\gamma_1 = 1$, $\gamma_2 = 0.34$, $y_1^1 = 1$, $y_2^1 = 1$, $y_3^1 = 1$, $y_4^1 = 0.53$, $y_5^1 = 0.24$, $y_6^1 = 0.71$, $y_7^1 = 0.41$, $y_4^2 = 0.47$, $y_5^2 = 0.76$, $y_6^2 = 0.29$, $y_7^2 = 0.06$, $x_{(1,2)}^1 = 1$, $x_{(1,3)}^1 = 0.35$, $x_{(1,5)}^1 = 0.06$, $x_{(1,6)} = 0.59$, $x_{(2,3)}^1 = 0.76$, $x_{(2,6)}^1 = 0.24$, $x_{(3,4)}^1 = 0.88$, $x_{(4,6)}^1 = 0.18$, $x_{(5,7)}^1 = 0.42$, $x_{(6,7)}^1 = 0.41$, $x_{(4,5)}^2 = 0.93$, $x_{(5,6)}^2 = 0.53$, $x_{(5,7)}^2 = 0.06$, $x_{(6,7)}^2 = 0.06$. It is depicted in Figure 3.1:

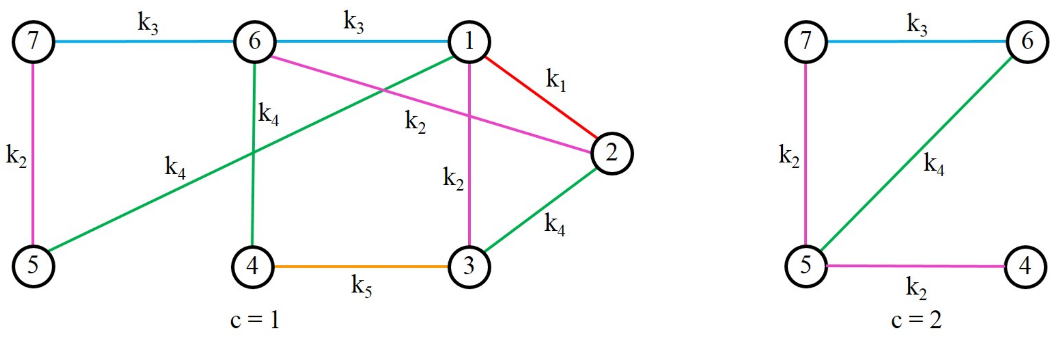


Figure 3.1: Solution where $\gamma_1 = 1$ and $\gamma_2 = 0.34$ and constraints (3.24) are violated by v_4^2 and v_5^2 in $c = 2$.

Proposition 3.2. *The constraints*

$$\sum_{c=1}^{\bar{c}} \left\{ x_e^c + \sum_{f \in \{\delta_h(u) \cup \delta_h(v)\}} x_f^c \right\} \leq 2 \quad e = (v,u) \in E : \ell(e) = k,\ h \in L \setminus \ell(e) \quad (3.26)$$

are valid for the RCCP.

Proof. These constraints impose that if an edge $e = (v, u)$ having color $\ell(e) = k$ is selected, then at most one edge having color $h \neq k$ and belonging to the set $\{\delta_h(v) \cup \delta_h(u)\}$ can be selected. □

Proposition 3.3. *The constraints*

$$\sum_{e \in \delta_k(v)} x_e^c \leq y_v^c \qquad k \in L,\ v \in V,\ c = 1, \ldots, \bar{c} \qquad (3.27)$$

are valid for the RCCP.

Proof. If a vertex v belongs to a cycle c, then at most one edge of color k and incident on v can be selected. Note that for fixed a vertex v, a cycle c and a color k, the inequality

$$\sum_{e \in \delta_k(v)} x_e^c \leq |\delta_k(v)| y_v^c \qquad k \in L,\ v \in V,\ c = 1, \ldots, \bar{c} \tag{3.28}$$

represents an aggregate version of constraints (3.6), when the color k is fixed. Moreover, due to (3.8) the left-hand side of (3.28) results in

$$\sum_{e \in \delta_k(v)} x_e^c \leq \sum_{e \in E_k} x_e^c \leq 1 \qquad k \in L,\ v \in V,\ c = 1, \ldots, \bar{c}, \tag{3.29}$$

and due to (3.5) all the edges x_e^c, i.e. $e \in \delta(v)$, are equal to 0 if y_v^c is equal to 0. Thanks to these two observations the right-hand side of (3.28) can be reduced to y_v^c, i.e. exactly constraints (3.27). □

Proposition 3.4. *The constraints*

$$\sum_{e \in \delta_k(v)} x_e^c - \sum_{e \in \{\delta(v) \setminus \delta_k(v)\}} x_e^c \leq 0 \qquad v \in V,\ k \in L,\ c = 1, \ldots, \bar{c} \tag{3.30}$$

are satisfied by all the optimal RCCP solutions.

Proof. These constraints impose that for each vertex v, for each color k and for each possible cycle $c = 1, \ldots, \bar{c}$, if an edge incident to v and having color k is selected, then an edge incident to v and having a different color must be selected. □

Proposition 3.5. *The valid inequalities (3.27) and (3.30) are equivalent.*

Proof. The valid inequalities (3.27) state that, fixed $k \in L$ and $c = \{1, \ldots, \bar{c}\}$,

$$\sum_{e \in \delta_k(v)} x_e^c \leq y_v^c$$

adding and subtracting $\frac{1}{2}\sum_{e\in\delta(v)} x_e^c$ to the left-hand side, we obtain

$$\sum_{e\in\delta_k(v)} x_e^c + \frac{1}{2}\sum_{e\in\delta(v)} x_e^c - \frac{1}{2}\sum_{e\in\delta(v)} x_e^c \leq y_v^c$$

which, due to (3.5), is equivalent to

$$\sum_{e\in\delta_k(v)} x_e^c - \frac{1}{2}\sum_{e\in\delta(v)} x_e^c \leq 0. \tag{3.31}$$

Note that, with easy mathematical operations, one can see that (3.31) are exactly the valid inequalities (3.30). One also observes that without constraints (3.5), constraints (3.27) are stronger than constraints (3.30). □

Proposition 3.6. *The constraints*

$$x_e^c - \sum_{h\in L\setminus\ell(\delta(v))}\ \sum_{f\in\delta_h(u)} x_f^c \leq 0 \qquad v:\ \zeta_v = 2,\ c = 1,\ldots,\bar{c} \tag{3.32}$$

are valid for the RCCP.

Proof. These constraints state that if an edge $e = (v, u)$, incident to a vertex v having colored degree $\zeta_v = 2$, is selected, then at least one edge that is incident on the vertex u and having color $h \in \{L \setminus \ell(\delta(v))\}$ must be selected. □

Proposition 3.7. *The constraints*

$$\sum_{\{v\in V:\delta_k(v)\neq\emptyset\}} \left\{ y_v^c - \sum_{e\in\delta_k(v)} x_e^c \right\} \geq 0 \qquad k \in L,\ c = 1,\ldots,\bar{c} \tag{3.33}$$

are valid for the RCCP.

Proof. These constraints impose that for each color k and for each cycle c, twice the number of edges having color k and belonging to c must be less than or equal to the number of vertices belonging to c on which those edges are incident, since on a single vertex cannot incide twice the same color. □

An example of a solution violating constraints (3.33) when $k = k_2$ but satisfying the constraints (3.2 - 3.11) is depicted in Figure (3.2):

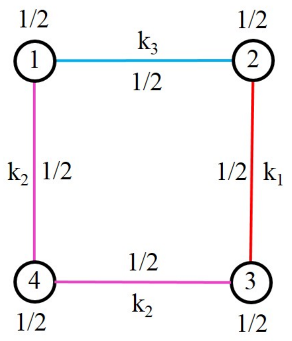


Figure 3.2: A constraint (3.33) is violated for $k = k_2$

Proposition 3.8. *Let $F = \{(v,u) \in E : |\ell(\delta(v)) \cup \ell(\delta(u))| = 3\}\}$. Moreover, for each edge $e = (u,v)$, let $\Delta(e) = \{\delta(u) \cup \delta(v)\}$ and let $\Delta_k(e) = \{\delta_k(v) \cup \delta_k(u)\}$. Then the constraints*

$$x^c_{(e)} - \sum_{f \in \Delta_k(e)} x^c_f \leq 0 \quad e = (u,v) \in F,\ k \in \{\ell(\Delta(e)) \setminus \ell(e)\},\ c = 1, \dots, \bar{c} \tag{3.34}$$

are valid for the RCCP.

Proof. These constraints impose that each edge $e = (u,v)$ such that $\big\{\{\ell(\delta(u)) \cup \ell(\delta(v))\} \setminus \ell(e)\big\} = \{h,k\}$ can belong to a *RCC* if and only if at least one edge of the set $\{\delta_h(u) \cup \delta_h(v)\}$ and one of the set $\{\delta_k(u) \cup \delta_k(v)\}$ is selected. It is clear that, since we are looking for a *RCC*, if an edge of the set F will be selected, then exactly one edge for each set will belong to the solution. □

We can extend the observation made for constraints (3.34) to the set of edges $\bar{F} = \{(u,v) \in E : |\ell(\delta(u)) \cup \ell(\delta(v))| = 4\}$.

Proposition 3.9. *The constraints*

$$x^c_e - \sum_{f \in \Delta_t(e) \cup \Delta_s(e)} x^c_f \leq 0 \quad e = (u,v) \in \bar{F},\ t,s \in \{\ell(\Delta(e)) \setminus \ell(e)\},\ c = 1, \dots, \bar{c} \tag{3.35}$$

are valid for the RCCP.

The last set of valid inequalities that we will present, differently from all the sets described until now, are not easy to identify. Let $P(v,u)$ denote a path between the vertices v and u, where $v \neq u$, and let $\mathcal{P}$ be the set of all the rainbow paths of the graph G, i.e. $\mathcal{P} = \{P(u,v) : P(u,v) \text{ is rainbow}\}$.

Proposition 3.10. *The constraints*

$$\sum_{c=1}^{\bar{c}} \left\{ \sum_{e \in \delta_k(v) \cup \delta_k(u)} x_e^c + \sum_{e \in P(u,v)} x_e^c \right\} \leq |P(u,v)| + 1 \quad k \in L \setminus \ell(P(u,v)),\ P(u,v) \in \mathcal{P} \tag{3.36}$$

are valid for the RCCP.

Proof. These constraints state that if all the edges of the rainbow path $P(v,u)$ belong to the solution, then at most one edge belonging to the set $\{\delta_k(v) \cup \delta_k(u)\}$, where $k \in L \setminus \ell(P(u,v))$, can be selected. Conversely, if two edges from the set $\{\delta_k(v) \cup \delta_k(u)\}$ are selected, then at least one edge of the rainbow path $P(v,u)$ cannot be in the optimal solution. □

In the Figure (3.3) is depicted an example of the structure described above for the valid inequalities (3.36). Note that the valid inequalities (3.26) are a particular

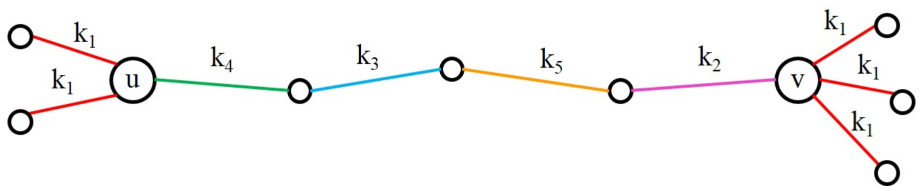


Figure 3.3: An example of valid inequalities (3.36)

case of this set of inequalities, in which the rainbow path is a single edge.

3.4 Branch-and-cut algorithm

In this section we describe the main features of the branch-and-cut algorithm for the Rainbow Cycle Cover Problem. The description of the steps is summarized in

Procedure 2. The first step of our algorithm consists in a preprocessing phase in which we identify all vertices and edges that satisfy the properties (3.15) and (3.16). We then set the corresponding variables to 0, which allows us to reduce the instance size. In line 2, the initial subproblem is obtained by relaxing constraints (3.6) and (3.7) as well as the integrality constraints (3.12), (3.13) and (3.14). Note that, thanks to constraints (3.5), constraints (3.6) are redundant in an integer solution, but we add them as valid cuts. We also add constraints (3.17), (3.18), (3.20), (3.21), (3.22) and (3.23) to the initial subproblem. From this point, in line 12 a search for violated constraints (3.7) is performed on the integer solutions, to prevent solutions with more than one cycle associated to each variable γ_c. Furthermore, in line 14, at non-integer solutions, a search for violate constraints (3.6) and (3.7) and violated valid inequalities (3.24), (3.25), (3.26), (3.27) and (3.32) is performed. Valid inequalities (3.33), (3.34) and (3.35) turned out to be ineffective and were not considered. A subset of the most violated inequalities of each type is added to the cut-pool. Moreover, since identifying all rainbow paths between all pairs of vertices in a graph is not possible, a search for violated inequalities (3.36) is performed only among the set $\mathcal{SP} = \{P(u,v) \in \mathcal{P} : P(u,v) \text{ is a shortest path}\}$. However, except for (3.7) and (3.36), in order to identify violated inequalities, we consider all of them and verify which are violated by the current relaxed solution. The algorithm for the identification of the most violated constraint (3.7) is a simple max-flow separation problem. In line 16, all the violated constraints are added to the model. If no violated constraints are identified, in line 18, branching is performed in priority on the y_v^c and x_v^c variables with the lower index c and having the fractional value closest to 0.5.

3.5 Computational results

The algorithm was coded in C and solved using IBM ILOG CPLEX 12.5. The computational experiments were performed on a 64-bit GNU/Linux operating system, 96 GB of RAM and one processor Intel Xeon X5675 running at 3.07 GHz. Experiments for the RCCP were conducted on randomly generated instances and their results are reported in Table 3.1. Each instance is characterized by the number of vertices n (size), the number of edges m and the number of colors l. For each

Procedure 2: Branch-and-cut algorithm

Input: an integer program P.
Output: an optimal solution of P, if exists.

```
1  $ub \leftarrow \infty$, $L = \emptyset$
2  Define a first subproblem $S_0$ and insert it in the list $L$
3  while $L$ is not empty do
4      choose a subproblem and remove it from $L$
5      solve the subproblem to obtain the solution $z$
6      if $z < ub$ then
7          if the solution is integer then
8              if the solution is feasible then
9                  $ub \leftarrow z$
10                 update incumbent solution
11             else
12                 search and add constraints 3.7 on integer solution
13         else
14             search violated constraints
15             if violated constraints are identified then
16                 add them to the model
17             else
18                 branch on a variable and add the corresponding subproblems
                   in $L$
```

instance with n vertices, the number of edges is set to $m = \lceil n(n-1)/2 \times d + n \rceil$, with $d \in \{0.1, 0.2, 0.3\}$, and the number of colors is set to $\lceil \log(m)/2 \rceil$, $\lceil \log(m) \rceil$ and $\lceil 2\log(m) \rceil$. For each size, there are nine different scenarios. We have generated five instances for each scenario, with the same number of nodes, edges and colors and the results reported in each line of our tables are average values over these five instances. In Table 3.1 the first four columns report the characteristics of each scenario: scenario ID, the number of vertices (n), the number of edges (m) and the number of colors (l), respectively. This table also provides the number of rainbow cycles (cycles), the number of non-trivial cycles and the number of trivial cycles (non-trvl and trvl, respectively) the value of the optimal solution (Obj), the computing time (t(s)) in seconds and the number of nodes in the search tree (nodes). We have imposed a time limit equal to $10,800$ seconds. Whenever α instances of a scenario was not solved to optimality within the time limit, we report (α) close to the solution value, therefore the value reported is an upper bound on the optimal solution. We also refer to the solutions with the symbol (α) as the *best known* solutions. Note that in the objective function, the part that minimizes the number of non-trivial cycles is less then or equal to $\bar{c}$ in the worst case. Thanks to this observation a value equal to $2\,\bar{c}$ is given to the weight M.
The results show that when the number of vertices increases, instances with a large number of edges and a small number of colors are the hardest to solve. Test results show that in these cases also the computational time and the number of nodes in the search tree seem to increase. This is due to the symmetry of the problem. Indeed several cycles can have the same number of colors and several equivalent solutions can be identified. The number of colors also affects the solution in terms of the presence of trivial cycles, mainly for the instances with a small number of edges.

3.5.1 LP lower bounds and duality gaps

We present the LP lower bounds and the duality gaps for the RCCP obtained by adding one valid inequality each time, respectively in Table 3.2 and Table 3.3. The first two columns of the Table 3.2 provide the instance ID and the objective value. A symbol * appears in the table close to the solution value to indicate that such

ID	Instance			RCCP					
	n	m	l	**cycles**	**non-trvl**	**trvl**	**Obj**	**t(s)**	**nodes**
1	20	39	3	18.00	1.00	17.00	120.00	0.01	0.00
2	20	39	6	13.60	2.00	11.60	83.20	0.14	0.00
3	20	39	11	10.60	2.20	8.40	61.00	0.18	0.00
4	20	58	3	14.80	2.60	12.20	88.00	0.13	0.00
5	20	58	6	9.80	3.40	6.40	48.20	1.76	5.60
6	20	58	12	6.80	2.20	4.60	34.40	1.71	12.00
7	20	77	4	10.00	3.80	6.20	47.20	2.41	21.20
8	20	77	7	6.80	3.60	3.20	26.00	5.86	77.00
9	20	77	13	4.80	2.60	2.20	18.00	5.71	65.60
10	30	74	4	21.80	3.20	18.60	207.80	2.59	4.00
11	30	74	7	19.40	2.80	16.60	185.40	15.16	16.80
12	30	74	13	15.40	2.60	12.80	143.40	9.33	8.60
13	30	117	4	18.20	4.80	13.40	152.20	21.38	38.60
14	30	117	7	13.00	4.40	8.60	99.00	56.52	114.80
15	30	117	14	9.60	3.20	6.40	73.60	54.07	119.20
16	30	161	4	15.20	6.00	9.20	107.20	180.50	541.20
17	30	161	8	8.00	5.20	2.80	36.00	210.39	344.00
18	30	161	15	5.40	3.60	1.80	23.40	55.79	67.80
19	40	118	4	30.60	3.60	27.00	381.60	10.67	22.00
20	40	118	7	24.80	3.80	21.00	297.80	129.83	49.80
21	40	118	14	20.40	3.40	17.00	241.40	34.08	10.40
22	40	196	4	25.00	6.20	18.80	269.40	314.22	191.20
23	40	196	8	15.80	6.00	9.80	143.20	361.94	281.60
24	40	196	16	11.00	4.00	7.00	102.00	480.33	457.60
25	40	274	5	14.60	8.60	6.00	(2)92.60	5136.55	2894.00
26	40	274	9	8.80	6.20	2.60	(2)42.60	6420.92	2141.20
27	40	274	17	5.20	3.80	1.40	(1)23.40	3185.14	1119.20
28	50	173	4	35.80	5.60	30.20	519.00	526.10	42.00
29	50	173	8	30.40	5.40	25.00	430.40	1185.35	368.40
30	50	173	15	23.40	5.20	18.20	314.60	694.63	357.40
31	50	295	5	25.40	8.00	17.40	(1)303.80	7354.71	2027.80
32	50	295	9	18.40	7.20	11.20	(1)197.60	5558.08	1477.40
33	50	295	17	12.80	5.60	7.20	128.00	2695.05	756.20
34	50	418	5	19.80	10.20	9.60	(5)173.40	10800.00	1194.20
35	50	418	9	12.80	8.20	4.60	(5)86.40	10214.81	1130.00
36	50	418	18	6.60	4.40	2.20	(3)41.80	8954.48	1247.00

Table 3.1: Summary of computational results for the RCCP

ID	RCCP							
	Obj		w(P)	w(P1)	w(P2)	w(P3)	w(P4)	w(P5)
1	120.00		120.00	120.00	120.00	120.00	120.00	120.00
2	83.20		79.62	81.14	79.71	79.70	79.62	79.62
3	61.00		56.33	56.99	56.38	56.45	56.38	56.51
4	88.00		85.38	86.17	85.98	85.38	85.38	85.79
5	48.20		29.76	32.87	30.50	29.82	30.12	30.86
6	34.40		26.05	27.06	26.84	26.36	26.12	26.16
7	47.20		28.86	32.37	30.99	28.88	28.99	29.69
8	26.00		16.92	18.10	17.43	17.06	17.02	16.92
9	18.00		11.63	13.08	12.92	12.04	11.63	11.63
10	207.80		191.20	194.43	193.09	191.20	191.84	192.98
11	185.40		131.39	142.10	134.91	131.43	131.91	132.50
12	143.40		122.14	123.89	122.67	122.31	124.18	122.43
13	152.20		94.83	105.95	105.21	94.83	96.71	97.65
14	99.00		62.67	71.05	67.90	62.73	64.81	65.31
15	73.60		53.60	58.47	54.51	53.93	53.78	54.16
16	107.20		44.66	52.90	52.95	44.66	46.04	50.17
17	36.00		17.58	18.93	22.11	17.71	17.67	17.70
18	23.40		16.01	20.59	18.10	16.50	16.59	16.01
19	381.60		320.46	326.28	326.17	320.46	322.99	324.15
20	297.80		235.51	245.40	244.61	235.59	236.46	240.75
21	241.40		202.22	211.60	207.59	202.57	203.82	206.17
22	269.40		133.77	164.04	155.76	133.77	138.69	144.44
23	143.20		77.03	88.87	91.21	77.06	82.55	78.44
24	102.00		69.79	75.49	74.25	70.00	69.87	69.79
25	92.60	*	31.74	39.58	36.03	31.78	34.37	31.74
26	42.60	*	23.89	26.67	26.67	23.89	26.67	23.89
27	23.40	*	19.08	21.87	19.08	19.08	19.08	19.08
28	519.00		388.47	412.81	410.57	388.47	393.28	407.81
29	430.40		277.13	308.62	296.86	277.33	282.80	288.49
30	314.60		233.08	251.27	243.86	233.56	236.17	233.20
31	303.80	*	148.93	176.66	164.85	148.95	153.38	155.18
32	197.60	*	114.41	132.04	124.42	114.45	116.63	114.41
33	128.00		97.06	108.35	101.71	97.40	97.64	97.91
34	173.40	*	63.98	78.23	79.88	64.02	67.21	67.07
35	86.40	*	51.16	55.10	52.84	51.16	51.16	52.84
36	41.80	*	40.06	40.06	40.06	40.06	40.06	40.06

Table 3.2: Linear programming lower bounds for RCCP

ID	RCCP					
	gP(%)	**gP1(%)**	**gP2(%)**	**gP3(%)**	**gP4(%)**	**gP5(%)**
1	0.0	0.0	0.0	0.0	0.0	0.0
2	4.3	2.5	4.2	4.2	4.3	4.3
3	7.7	6.6	7.6	7.5	7.6	7.4
4	3.0	2.1	2.3	3.0	3.0	2.5
5	38.3	31.8	36.7	38.1	37.5	36.0
6	24.3	21.3	22.0	23.4	24.1	24.0
7	38.9	31.4	34.3	38.8	38.6	37.1
8	34.9	30.4	32.9	34.4	34.6	34.9
9	35.4	27.4	28.2	33.1	35.4	35.4
10	8.0	6.4	7.1	8.0	7.7	7.1
11	29.1	23.4	27.2	29.1	28.9	28.5
12	14.8	13.6	14.5	14.7	13.4	14.6
13	37.7	30.4	30.9	37.7	36.5	35.8
14	36.7	28.2	31.4	36.6	34.5	34.0
15	27.2	20.6	25.9	26.7	26.9	26.4
16	58.3	50.7	50.6	58.3	57.1	53.2
17	51.2	47.4	38.6	50.8	50.9	50.8
18	31.6	12.0	22.6	29.5	29.1	31.6
19	16.0	14.5	14.5	16.0	15.4	15.1
20	20.9	17.6	17.9	20.9	20.6	19.2
21	16.2	12.3	14.0	16.1	15.6	14.6
22	50.3	39.1	42.2	50.3	48.5	46.4
23	46.2	37.9	36.3	46.2	42.4	45.2
24	31.6	26.0	27.2	31.4	31.5	31.6
25	65.7	57.3	61.1	65.7	62.9	65.7
26	43.9	37.4	37.4	43.9	37.4	43.9
27	18.5	6.5	18.5	18.5	18.5	18.5
28	25.2	20.5	20.9	25.2	24.2	21.4
29	35.6	28.3	31.0	35.6	34.3	33.0
30	25.9	20.1	22.5	25.8	24.9	25.9
31	51.0	41.9	45.7	51.0	49.5	48.9
32	42.1	33.2	37.0	42.1	41.0	42.1
33	24.2	15.3	20.5	23.9	23.7	23.5
34	63.1	54.9	53.9	63.1	61.2	61.3
35	40.8	36.2	38.8	40.8	40.8	38.8
36	4.2	4.2	4.2	4.2	4.2	4.2

Table 3.3: Linear programming duality gap for RCCP

value is an upper bound of the optimal solution value. The remaining columns provide lower bounds $w(P)$, $w(P1)$, $w(P2)$, $w(P3)$, $w(P4)$ and $w(P5)$, where P denotes the polytope obtained by relaxing the integrality constraints, while $P1$, $P2$, $P3$, $P4$ and $P5$ denote the intersection of P with (3.27), (3.26), (3.24) – (3.25), (3.36) and (3.32), respectively. Table 3.3 provide the duality gap obtained on the six polytopes comparing with the optimal solution. From the tables we can observe that the valid inequalities often help to improve the lower bound $w(P)$ and that the best lower bounds are provided by $w(P1)$ and $w(P2)$. However, it is interesting to observe that for the hardest instances, the values of the gap are significantly high. The presence of the constant M in the objective function affects the results, but symmetry seems to remain the main problem.

Chapter 4

The Rainbow Spanning Forest Problem

“ *A falling tree makes more noise than a growing forest.* ”

old saying

4.1 Introduction and Problem Description

The purpose of this chapter is to present *The Rainbow Spanning Forest Problem* (RSFP), another example of spanning tree problem defined on a connected undirected edge-colored graph.

Let $G = (V, E, L)$ be a connected and undirected graph, where V is the set of n vertices, E is the set of m edges and L a set of l colors. In addition, let $\ell : E \to L$ be a coloring function that assigns to each edge $e \in E$ a color $\ell(e)$ from the set L. A *rainbow spanning forest* of G is a spanning forest $F = (V_F, E_F, L_F)$ of the graph G, with $V_F = V$, $E_F \subseteq E$ and $L_F \subseteq L$, such that all components are rainbow. A component of the forest is a connected acyclic graph, therefore a *rainbow component* of F is a tree $T = (V_T, E_T, L_T)$, where $V_T \subseteq V$ and $E_T \subseteq E_F$, in which all edges have different colors, i.e. $|L_T| = |\ell(E_T)| = |E_T|$. Note that if F has $\bar{c}$ rainbow components $T_1, \ldots, T_{\bar{c}}$, then $V_F = V = \cup_{i=1}^{\bar{c}} V_{T_i}$, $E_F = \cup_{i=1}^{\bar{c}} E_{T_i}$ and $L_F = \cup_{i=1}^{\bar{c}} L_{T_i}$. The

Rainbow Spanning Forest Problem (RSFP) consists of finding a rainbow spanning forest with the least number of rainbow trees. The RSFP on general graphs is known to be NP-hard [34]. The remainder of this chapter is organized as follows. In Section 4.2 we prove that the RSFP is NP-hard on trees and we also provide an alternative proof of the NP-hardness of the problem on general graphs. Moreover, in Section 4.3 we introduce two new integer mathematical formulations (ILP1) and (ILP2) and for the second one we propose valid inequalities. A greedy algorithm and a multi-start scheme applied to the greedy algorithm to improve its results are presented in Section 4.4. Section 4.5 contains the computational results.

4.2 Problem Complexity

In this section we provide an alternative proof of the $\mathcal{NP}$-hardness of the problem on general edge-colored graphs. Moreover, we prove that the RSFP is $\mathcal{NP}$-hard on edge-colored trees. To the best of our knowledge, no proof for the $\mathcal{NP}$-hardness of the RSFP on trees has ever been put forward. A brief subsection is dedicated to a polynomial case.

The decision version of the RSFP, namely, the *Bounded Rainbow Spanning Forest Problem* (BRSFP) is as follows:

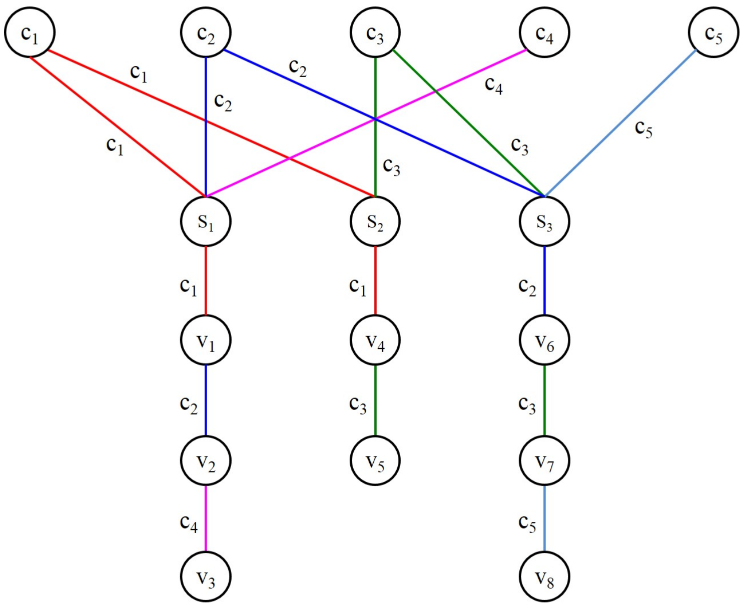
Definition 4.1. ***BRSFP:*** *Given a connected, undirected and edge-colored graph G and a positive integer z: is there a rainbow spanning forest such that the number of components is less than or equal to z?*

Theorem 4.1. *The BRSFP on general edge-colored graphs is $\mathcal{NP}$-Complete.*

Proof. We prove the theorem by reduction from the Minimum Set Covering Problem (MSC). Let $C = \{c_1, ..., c_d\}$ be a set of d elements, $S = \{S_1, ..., S_k\}$ be a family of k subsets of C, i.e. $S_i \subseteq C$, $i = 1, ..., k$, and s be a positive number. The decisional version of MSC consists in selecting no more than s subsets in S that cover all the elements of C.
We define, from the generic instance of MSC, a graph $G = (V, E, L)$, with a coloring function of the edges, where $V = \{c_1, ..., c_d, S_1, ..., S_k\} \cup \{v_j : j = 1, ..., b\}$, with $b = \sum_{t=1}^{k} |S_t|$. Moreover we define the set E in the following way:

C	S
c_1	$S_1=\{c_1, c_2, c_4\}$
c_2	$S_2=\{c_1, c_3\}$
c_3	$S_3=\{c_2, c_3, c_5\}$
c_4	
c_5	

(a)



(b)

Figure 4.1: (a) A generic instance of the Minimum Set Covering Problem, and, (b) the corresponding instance of the Bounded Rainbow Spanning Problem.

- $(c_h, S_t) \in E$ if and only if $c_h \in S_t$. We associate to this edge the color c_h;
- $\forall S_t$ we build a path $P_t = \{S_t, v_{t_1}, \ldots, v_{t_{|S_t|}}\}$ of $|S_t|$ edges. We associate to the i^{th} edge of the path P_t the color of the i^{th} element of S_t.

Therefore in G there are $d + k + b$ vertices and $2b$ edges. This construction can be made in polynomial time (see the example in Figure 4.1). We want to show that there exists a covering of C with at most s subsets if and only if there exists a spanning forest of G with at most $s + k$ rainbow components.
Consider now a cover of C having size s, that is a selection of s ordered sets $\{S_{t_1}, \ldots, S_{t_s}\}$ such that their union is C. We can define a spanning forest with at most $s + k$ rainbow components by selecting the edges that connect the vertex S_{t_j} to the vertices $\{c_{j_1}, ..., c_{j_q}\}$ if the elements $c_{j_1}, ..., c_{j_q}$ belong to the set S_{t_j}. If an element c_h is covered by more than one set S_{t_j}, we select only one of the edges connecting c_h. Moreover, if we suppose that $c_{j_1}, ..., c_{j_q}$ are considered in the order in which they appear in the set S_{t_j}, to preserve the rainbow property, the path P_{t_j}, or part of the path, has to be disconnected. Two components are therefore created.

Since there are s vertices S_{t_j} that are connect to some vertex c_h, having assumed that there exists a covering of C with at most s subsets, then we disconnect s paths thus obtaining $2s$ components. For the remaining $k - s$ sets S_t, since they are not used for the coverage of C, we do not select the edges that connect them to some vertex c_h and therefore we can select all the edges of the corresponding path P_t. There are exactly $k - s$ paths that are not disconnected. Overall, the number of rainbow components is $2s + (k - s) = s + k$.
Conversely, suppose there exists a spanning forest with $s + k$ rainbow components. Of course the vertices $S_1, ..., S_k$ are in k different components, because they can only be connected using edges of the same color. Moreover, if we denote by H_t the component containing the vertex S_t, it is easy to observe that if H_t contains at least one edge (c_h, S_t), it cannot contain all the edges of the path P_t. Therefore, besides H_t another component exists, which consists of a part of the path P_t or of the entire path. Having assumed that there are $s + k$ rainbow components, then the components containing at least one edge (c_h, S_t) are s. This s components identify a covering of C of size s. □

Theorem 4.2. The RSFP on edge-colored trees is $\mathcal{NP}$-Complete.

Proof. We prove the theorem by reduction from the 3-SAT Problem. Let ϕ be our formula for 3-SAT, written in a conjunctive normal form, containing d literals $U = \{u_1, \ldots, u_d\}$ and b clauses $C = \{c_1, \ldots, c_b\}$. The decisional version of 3-SAT consists in verifying whether there exists an assignment of values to U that makes every clause true. We now define, from the generic instance of 3-SAT, an acyclic graph $T = (V, E, L)$, with a coloring function of the edges (see the example in Figure 4.2). At the beginning let the set of the vertices be $V = \{r\}$, where r is the root of the graph T, and let E be the empty set. To each $c_h \in C$ we associate a vertex v_{c_h} and define the edge $(r, v_{c_h}) \in E$ of color c_h. Moreover, for each $c_h \in C$ we define three vertices h_{1,u_i}, h_{2,u_j} and h_{3,u_k}, where u_i, u_j and u_k are the literals of clause c_h, and three edges $(v_{c_h}, h_{1,u_i}), (v_{c_h}, h_{2,u_j}), (v_{c_h}, h_{3,u_k})$ to which we associate the same color h. Note that the edge (v_{c_h}, h_{i,u_t}) is associated with the i^{th} literal of the clause c_h. Furthermore for all h_{i,u_t}, we build in the graph T a path $P_{h_{i,u_t}}$, whose first vertex is h_{i,u_t}, as follows:

- $P_{h_{i,u_t}}$ has length $|N_t|$ if the clause c_h contains u_t

- $P_{h_{i,u_t}}$ has length $|Y_t|$ if the clause c_h contains $\neg u_t$,

where, for each $u_t \in U$, if u_t is in the position $\bar{p}$ of a clause $\bar{c}$, then the pair $(\bar{p}, \bar{c})$ belongs to Y_t. Otherwise, if $\neg u_t$ is in the position $\bar{p}$ of a clause $\bar{c}$, then the pair $(\bar{p}, \bar{c})$ belongs to N_t. For instance, in Figure 4.2, for literal u_1 we have $Y_1 = \{(1,1),(1,2)\}$ and $N_1 = \{(1,3)\}$. More in detail, for each $u_t \in U$ and for each $((y^1, y^2),(n^1, n^2))$, i.e. $(y^1, y^2) \in Y_t$ and $(n^1, n^2) \in N_t$, we add an edge e to the path $P_{{y^2}_{y^1,u_t}}$ and an edge f to the path $P_{{n^2}_{n^1,u_t}}$. To e and f we assign a color a, different from all the colors used until now. Since $|Y_1| = 2$ and $|N_1| = 1$, we have two pairs associated to u_1: $((1,1),(1,3))$ and $((1,2),(1,3))$. For $((1,1),(1,3))$, in Figure 4.2, we have $(1_{1,u_1}, w_1)$ and $(3_{1,u_1}, w_2)$ in $P_{1_{1,u_1}}$ and $P_{3_{1,u_1}}$, respectively, having color 4. Moreover, for $((1,2),(1,3))$ we have $(2_{1,u_1}, w_3)$ and (w_2, w_4) in $P_{2_{1,u_1}}$ and $P_{3_{1,u_1}}$, respectively, having color 5. Note that since in a same clause cannot be present a literal u_t and its negated $\neg u_t$, each path $P_{h_{i,u_t}}$ is always rainbow. Therefore the set of vertices, edges and colors of the tree T are the following:

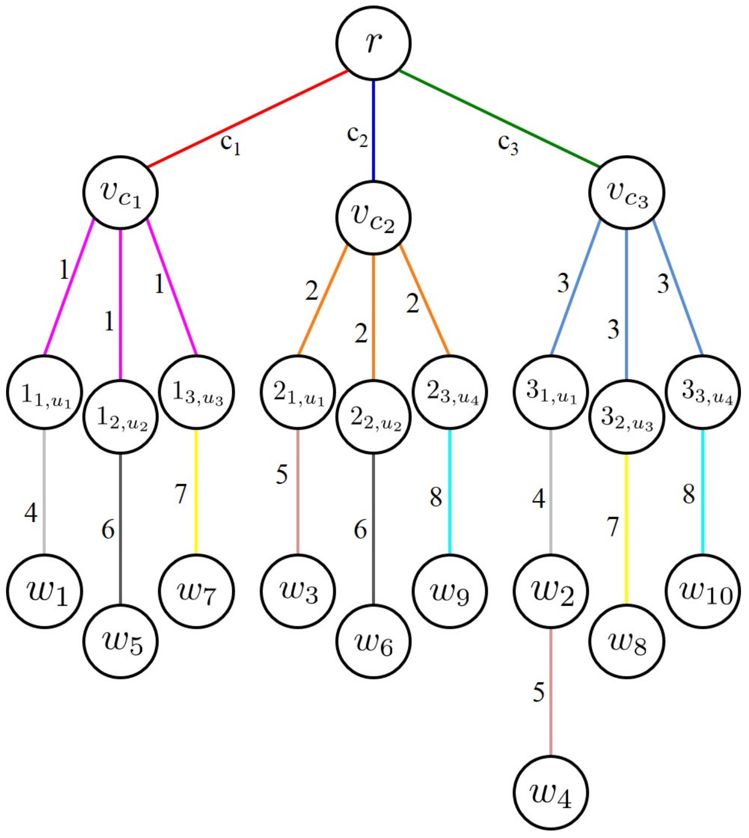
- $V = \{r\} \cup \{v_{c_h}, h_{1,u_i}, h_{2,u_j}, h_{3,u_k} : h = 1, \ldots, b\} \cup \{w_i : i = 1, \ldots, 2\bar{q}\}$,
- $E = \{(r, v_{c_h}), (v_{c_h}, h_{1,u_i}), (v_{c_h}, h_{2,u_j}), (v_{c_h}, h_{3,u_k}) : h = 1, \ldots, b\} \cup \{e_i : i = 1, \ldots, 2\bar{q}\}$,
- $L = \{c_h, h : h = 1, \ldots, b\} \cup \{i : i = 1, \ldots, \bar{q}\}$,

where $\bar{q} = \sum_{t=1}^{d} |Y_t| \times |N_t|$. This construction can be accomplished in polynomial time.

We want to show that there is an assignment of values to U that makes every clause true if and only if exists a spanning forest of T using $2b + 1$ rainbow components. Note that, in order to preserve the rainbow property, at most one of the three edges (v_{c_h}, h_{1,u_i}), (v_{c_h}, h_{2,u_j}), (v_{c_h}, h_{3,u_k}), associated with each clause c_h, for all $h \in \{1, \ldots, b\}$, can appear in a rainbow spanning forest (the three edges are incident to the same vertex v_{c_h} and have the same color h). Consider now an assignment of values to U which makes every clause true. We can define a rainbow spanning forest with $2b + 1$ components by selecting the edges $\{(r, v_{c_h}) : h \in \{1, \ldots, b\}\}$, whose colors are all different. Furthermore, for each clause c_h, among (v_{c_h}, h_{1,u_i}), (v_{c_h}, h_{2,u_j}), (v_{c_h}, h_{3,u_k}), we select the edge associated with the literal having value

$$(u_1 \vee u_2 \vee \neg u_3) \wedge (u_1 \vee \neg u_2 \vee u_4) \wedge (\neg u_1 \vee u_3 \vee \neg u_4)$$

(a)



(b)

Figure 4.2: (a) A generic instance of the 3-SAT Problem, and, (b) the corresponding instance of the Bounded Rainbow Spanning Forest Problem in Edge-Colored Trees.

true in c_h. If more than one literal is true, we arbitrarily select only one of the corresponding edges. Moreover, we select all the edges of the rainbow path $P_{h_{i,u_t}}$, for all h_{i,u_t}. Note that two edges belonging to the rainbow paths have the same color if and only if they are associated with pairs of literals (L_1, L_2) such that if $L_1 = u_t$, then $L_2 = \neg u_t$, which surely cannot be simultaneously true. Therefore, at least one of the two edges linking these paths to the vertices associated to the clauses containing the literals, does not belong to the rainbow spanning forest. This ensures that the two edges belong to different rainbow components. In total, we do not select $2b$ edges and therefore we obtain a rainbow spanning forest with

$2b+1$ components.
Conversely, suppose that there exists a spanning forest with $2b+1$ rainbow components. As previously observed, edges (v_{c_h}, h_{1,u_i}), (v_{c_h}, h_{2,u_j}), and (v_{c_h}, h_{3,u_k}), $h \in \{1,\dots,b\}$, have the same color and are incident to the same vertex v_{c_h}, therefore at most one of them can appear in the rainbow spanning forest. Moreover, since we have supposed that exists a spanning forest with $2b+1$ rainbow components, we are sure that exactly one of them has to appear in the rainbow spanning forest, otherwise it would be impossible to have the $2b+1$ components. The ones that appear in the rainbow spanning forest with $2b+1$ components represent an assignment of values to U, which makes every clause true. □

4.2.1 Problem Complexity: A Polynomial Case

In this section we want to prove that the RSFP is polynomially solvable when the optimal solution is a tree, namely when the graph contains a rainbow spanning tree. Obviously, it is not possible to reach this goal by enumerating all the spanning trees of G. This is because the algorithms that enumerate all the spanning tree of G are pseudo-polynomial [29].

Given a spanning tree T of G, it is a *maximum tree* of G if and only if $|L_T|$ is maximum. The following theorem holds [5]:

Theorem 4.3. *The problem of finding a maximum tree T in G is solvable in polynomial time.*

In particular, the algorithm of Broersma and Li [5] computes the maximum tree of G in $O(n^2m)$ time.

Theorem 4.4. *RSFP is solvable in polynomial time if there exists in G a rainbow spanning tree.*

Proof. Given a graph G, let T' be the maximum tree of G computed by the algorithm proposed by Broersma and Li. It is easy to see that if $|L_{T'}| = n-1$, then T' is a rainbow spanning tree of G. □

4.3 Mathematical Formulations

In this section we provide two different mathematical formulations for the RSFP. The two formulations use the following proposition:

Proposition 4.1. *Let $G = (V, E)$ be a graph on n vertices, and let $F = (V, E_F)$ be a spanning forest of G. Then, $0 \leq |E_F| \leq n - 1$. Furthermore, if $|E_F| = n - k$, then F has k components. In particular, F is a spanning tree if and only if it contains $n - 1$ edges.*

Therefore, it is easy to observe that given a graph G, a spanning forest of G with the least number of rainbow components is a rainbow spanning forest with the maximum number of edges.

4.3.1 First Mathematical Formulation

In this section we present a first integer linear mathematical formulation for the RSFP. We introduce the set of binary variables α_c, for $c = 1, \ldots, \bar{c}$, associated with each component c of a rainbow spanning forest, whose value is equal to 1 if and only if c contains at least one vertex. We define binary variables y_v^c equal to 1 if and only if vertex v belongs to component c, and binary variables x_e^c equal to 1 if and only if edge e belongs to component c. Note that the number of variables depends on the number of possible components, so it is useful to have a good upper bound on the optimal value. If we do not know an upper bound, we set $\bar{c} = n - 1$, that is equal to the maximum number of rainbow components that we can identify. In order to introduce constraints that can help prevent equivalent solutions, we define an index set $I_q = \{1, \ldots, q\}$, for an integer q, and the vertex set $V = I_n$. The formulation (ILP1) is then as follows:

$$\text{minimize } z = \sum_{c=1}^{\bar{c}} \alpha_c \tag{4.1}$$

subject to

$$\sum_{v \in V} y_v^c \leq (l + 1)\, \alpha_c \qquad c = 1, \ldots, \bar{c} \tag{4.2}$$

$$\alpha_c \leq \sum_{v \in V} y_v^c \qquad c = 1, \ldots, \bar{c} \tag{4.3}$$

$$\sum_{c=1}^{\bar{c}} y_v^c = 1 \qquad v \in V \tag{4.4}$$

$$x_e^c \leq y_v^c \qquad v \in V,\ e \in \delta(v) \tag{4.5}$$

$$\sum_{e \in E_k} x_e^c \leq \alpha_c \qquad c = 1, \ldots, \bar{c},\ k \in L \tag{4.6}$$

$$\sum_{c=1}^{\bar{c}} \sum_{e \in E(S)} x_e^c \leq |S| - 1, \qquad S \subseteq V,\ |S| \geq 2 \tag{4.7}$$

$$\sum_{e \in E} x_e^c = \sum_{v \in V} y_v^c - \alpha_c \qquad c = 1, \ldots, \bar{c} \tag{4.8}$$

$$\alpha_{c+1} \leq \alpha_c \qquad c = 1, \ldots, \bar{c} - 1 \tag{4.9}$$

$$y_1^1 = 1 \tag{4.10}$$

$$y_v^c \leq \sum_{w < v} y_w^{c-1} \qquad v \in V \setminus \{1\},\ c = 3, \ldots, \bar{c} \tag{4.11}$$

$$\alpha_c \in \{0, 1\} \qquad c = 1, \ldots, \bar{c} \tag{4.12}$$

$$y_v^c \in \{0, 1\} \qquad v \in V,\ c = 1, \ldots, \bar{c} \tag{4.13}$$

$$x_e^c \in \{0, 1\} \qquad e \in E,\ c = 1, \ldots, \bar{c}, \tag{4.14}$$

where $\delta(v)$ denotes the set of edges incident to v in G and $E_k = \{e \in E : \ell(e) = k\}$. The objective function (4.1) requires the minimization of the number of rainbow components. Constraints (4.2) and (4.3) are logical constraints linking the binary variables α_c with the binary variables y_v^c. Note that the maximum number of vertices that can belong to the same component is $l + 1$ since l is the number of different colors of the graph. Constraints (4.4) ensure that each vertex belongs to exactly one component. Constraints (4.5) impose that if a vertex is not in the component c, then the edges incident to that vertex cannot belong to the same tree. Constraints (4.6) impose that a component cannot contain two edges having the same color, ensuring the rainbow property. Constraints (4.7) are the subtour elimination constraints, introduced by Dantzig et al. [15], adapted to our variables. Constraints (4.8) impose that each component is a tree. They guarantee solutions with no more than one tree associated to each variable α_c. Constraints (4.9), (4.10) and (4.11) help break symmetries. Constraints (4.9) mean that there will never be a variable α_{c+1} equal to one if α_c is equal to zero, for any c. The constraints (4.10) and (4.11) are the symmetry breaking constraints introduced by Fischetti et al. [18] in the context of the Vehicle Routing Problem. Vertex v can belong to a component of index c if and only if at least one vertex w with a lower index belongs to the component of index $c-1$. For the resolution of the model we also use the following constraints:

$$y_v^c \leq \alpha_c \qquad v \in V,\ c = 1, \dots, \bar{c} \tag{4.15}$$

$$\sum_{e \in \delta_k(v)} x_e^c \leq y_v^c \qquad v \in V,\ c = 1, \dots, \bar{c},\ k \in L \tag{4.16}$$

$$\sum_{c=1}^{\bar{c}} \left\{ x_e^c + \sum_{f \in \{\delta_k(u) \cup \delta_k(v)\}} x_f^c \right\} \leq 2 \qquad e = (v,u) \in E,\ k \in L \setminus \{\ell(e)\} \tag{4.17}$$

proposed by Silvestri et al. [51] to solve the Rainbow Cycle Cover Problem, and which are valid for the RSFP. Constraints (4.15) state that if a vertex belongs to a component, then the variable representing that component must be used. The valid inequalities (4.16) impose that if vertex v belongs to a tree c, then at most one edge having color k and incident to v can be selected. Constraints (4.17) impose that if edge $e = (v,u)$ is selected, then at most one edge having color $k \neq \ell(e)$ and belonging to the set $\{\delta_k(v) \cup \delta_k(u)\}$ can be selected.

4.3.2 Second Mathematical Formulation

In this section we present a second integer linear mathematical formulation for the RSFP. Let x_e be a binary variable equal to 1 if and only if edge $e \in E$ belongs to the rainbow spanning forest. Before showing the formulation we need the following proposition:

Proposition 4.2. *Let E_k, $k \in L$, be the set of edges of the graph having color k, i.e. $E_k = \{e \in E : \ell(e) = k\}$, then we can write $E = \cup_{k \in L} E_k$. Moreover, let n_S be the number of components in S, $S \subseteq V$, $|S| \geq 2$, then it is easy to observe that*

$$\sum_{e \in E_k(S)} x_e \leq n_S$$

where $E_k(S) = E_k \cap E(S)$, and

$$n_s = |S| - \sum_{e \in E(S)} x_e$$

are valid for any rainbow spanning forest, and hence

$$\sum_{e \in E_k(S)} x_e \leq |S| - \sum_{e \in E(S)} x_e$$

that is

$$\sum_{e \in E_k(S)} x_e + \sum_{e \in E(S)} x_e \leq |S| \quad S \subseteq V,\ |S| \geq 2,\ k \in L \tag{4.18}$$

are valid for the RSPF and ensure the rainbow property.

The (ILP2) formulation is then as follows:

$$z = \max \sum_{e \in E} x_e \tag{4.19}$$

$$\sum_{e \in E} x_e \leq |S| - 1 \qquad S \subseteq V,\ |S| \geq 2 \tag{4.20}$$

$$\sum_{e \in E_k(S)} x_e + \sum_{e \in E(S)} x_e \leq |S| \qquad S \subseteq V,\ |S| \geq 2,\ k \in L \tag{4.21}$$

$$x_e \in \{0, 1\} \qquad e \in E. \tag{4.22}$$

The objective function 4.19 requires the maximization of the edges selected which, thanks to Proposition 4.1, ensures a rainbow spanning forest with the least number of components. Constraints 4.20 are the subtour elimination constraints proposed by Dantzig, Fulkerson and Johnson [15]. Constraints 4.21 ensure the rainbow property, as shown in Proposition 4.2.
If a weight w_e is associated to the edges, the objective function (4.19) becomes

$$z = \max \sum_{e \in E} w_e x_e. \tag{4.23}$$

Theorem 4.5. *Constraints (4.21) and (4.22) define the set of feasible solutions for the RSFP, i.e. constraints (4.20) are redundant.*

Proof. By summing constraints (4.21) on all colors $k \in L$ we obtain

$$\sum_{e \in E(S)} x_e + l \sum_{e \in E(S)} x_e \leq l|S| \qquad\qquad S \subseteq V,\ |S| \geq 2,$$

and with simple algebraic manipulations

$$\sum_{e \in E(S)} x_e \leq |S| - \frac{1}{l+1}|S| \qquad\qquad S \subseteq V,\ |S| \geq 2.$$

We distinguish two cases:

- if $|S| > l+1$, then

  $$|S| - \frac{1}{l+1}|S| < |S| - 1$$

  and hence

  $$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad\qquad S \subseteq V,\ |S| > l+1,$$

- if $|S| \leq l+1$, then

  $$|S| - \frac{1}{l+1}|S| = |S| - \epsilon \qquad\qquad 0 < \epsilon \leq 1$$

  but since $\sum_{e \in E(S)} x_e$ is a sum of integers, it is an integer, which implies that

  $$\sum_{e \in E(S)} x_e \leq \lfloor |S| - \epsilon \rfloor$$

  and therefore

  $$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad\qquad S \subseteq V,\ |S| \leq l+1.$$

$\square$

Note that if we ignore the integrality restrictions on variables x_e, $e \in E$, Theorem 4.5 is no longer verified. Figure 4.3 provides an example of non-integer solution

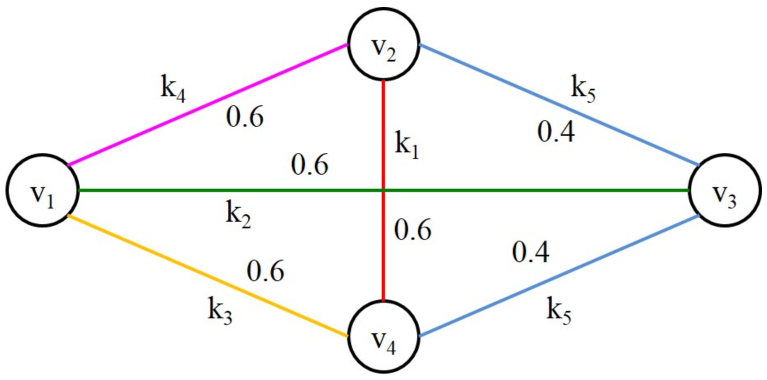


Figure 4.3: Non-integer solution.

that does not verify the theorem. It is easy to see that for $S = V$, constraints (4.21) are verified for all $k \in L$, while constraints (4.20) are not satisfied by the non-integer solution.

4.3.2.1 Valid inequalities for ILP2

This section is dedicated to some valid inequalities for the RSFP.

Given a tree $T = (V_T, E_T, L_T)$, where $V_T \subseteq V$, $E_T \subseteq E$ and $L_T \subseteq L$, we can write the set of edges E_T as follows

$$E_T = E_{T_L} \cup E_{T_I}$$

where $E_{T_L} = \{(u, v) \in E_T : |\delta_T(u)| = 1 \text{ or } |\delta_T(v)| = 1\}$ and $E_{T_I} = \{(u, v) \in E_T : |\delta_T(u)| > 1 \text{ and } |\delta_T(v)| > 1\}$.

Proposition 4.3. *Let $\bar{T}$ be a tree of G such that*

- *$E_{\bar{T}_I}$ is rainbow,*
- *$|\ell(E_{\bar{T}_L})| = 1$,*

- $\ell(E_{\bar{T}_I}) \cap \ell(E_{\bar{T}_L}) = \emptyset$,

then

$$\sum_{e \in E_{\bar{T}}} x_e \leq |V_{\bar{T}}| - |E_{\bar{T}_L}| \tag{4.24}$$

is valid for the RSFP.

Proof. It is easy to observe that $E_{\bar{T}} = E_{\bar{T}_L} \cup E_{\bar{T}_I}$ and that $|V_{\bar{T}}| = |E_{\bar{T}_L}| + |E_{\bar{T}_I}| + 1$, therefore the (4.24) becomes

$$\sum_{e \in E_{\bar{T}_L}} x_e + \sum_{e \in E_{\bar{T}_I}} x_e \leq |E_{\bar{T}_I}| + 1.$$

Let us suppose by contradiction that there exists a feasible solution $\bar{x}$ and a tree $\bar{T}$ such that

$$\sum_{e \in E_{\bar{T}_L}} \bar{x}_e + \sum_{e \in E_{\bar{T}_I}} \bar{x}_e > |E_{\bar{T}_I}| + 1,$$

If we denote by β the term $\sum_{e \in E_{\bar{T}}(I)} \bar{x}_e$, it is easy to observe that $|E_{\bar{T}_I}| + 1 - \beta$ represents the number n_{RC} of rainbow components that we obtain by selecting β edges in the rainbow subtree $(V_{\bar{T}_I}, E_{\bar{T}_I}, L_{\bar{T}_I})$. Therefore

$$\sum_{e \in E_{\bar{T}_L}} \bar{x}_e > |E_{\bar{T}_I}(I)| + 1 - \beta = n_{RC}$$

and then

$$\sum_{e \in E_{\bar{T}_L}} \bar{x}_e > n_{RC}$$

which is impossible because we have supposed that $\bar{x}$ is a feasible solution for the RSFP. □

It is easy to see that the subgraph $(\{v\}, \delta_k(v), \{k\})$, for a given vertex v and a color k, is a tree (see Figure 4.4(a)) such that

- $\delta_k(v)_I = \emptyset$,
- $|\delta_k(v)_L| = 1$,
- $\ell(\delta_k(v)_I) \cap \ell(\delta_k(v)_L) = \emptyset \cap \{k\} = \emptyset$,

therefore it satisfies all properties described in Proposition 4.3 and hence

$$\sum_{e \in \delta_k(v)} x_e \leq 1 \qquad\qquad v \in V, k \in L \tag{4.25}$$

are a particular case of (4.24), valid for the RSFP. From now on, we will refer to these as the *star inequalities*.

The constraints

$$x_e + \sum_{f \in \{\delta_h(v) \cup \delta_h(u)\}} x_f \leq 2 \qquad e = (v,u) \in E, \ell(e) = k, h \in L \setminus \{k\} \tag{4.26}$$

are another particular case of (4.24), valid for the RSFP. In particular (see Figure 4.4(b))

- e represents the internal rainbow set such that $\ell(e) = k$,
- h is the color of all the edges incident to leaf vertices,
- $k \neq h$ and therefore $\{k\} \cap \{h\} = \emptyset$.

From now on, we will refer to these as the *flake inequalities*.
Note that (4.25) and (4.26) are the same constraints proposed by Silvestri et al. [51] to solve the Rainbow Cycle Cover Problem, adapted to the ILP2 formulation.

Proposition 4.4. *Let $R = (V_R, E_R, L_R)$ be a rainbow cycle of cardinality three, i.e. $V_R = \{a,b,c\}$, $E_R = \{e_{ab}, e_{bc}, e_{ca}\}$ and $L_R = \{k_1, k_2, k_3\}$, and let $\Delta_h(a,b,c) = \{\delta_h(a) \cup \delta_h(b) \cup \delta_h(c)\}$, $h \in L$. The constraints*

$$x_{e_{ab}} + x_{e_{bc}} + x_{e_{ca}} + \sum_{f \in \Delta_h(a,b,c)} x_f \leq 3 \qquad\qquad h \notin L_R \tag{4.27}$$

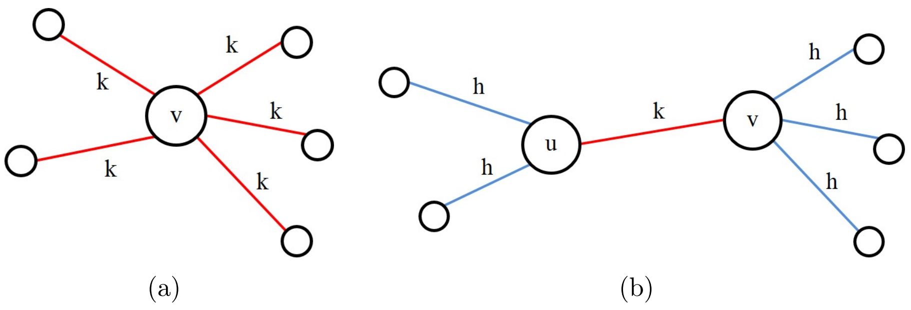


Figure 4.4: (a) An example of valid inequalities 4.25, and, (b) an example of valid inequalities 4.26.

are valid for the RSFP.
Moreover, let $h \in L_R$, we can suppose $h = \ell(e_{ab}) = k_1$, then

$$2x_{e_{ab}} + x_{e_{bc}} + x_{e_{ca}} + \sum_{f \in \Delta_{k_1}(a,b,c)\setminus\{e_{ab}\}} x_f \leq 3 \tag{4.28}$$

is valid for the RSFP.

Proof. Constraints (4.27) state that if at least one edge having color $h \notin L_R$ is selected, the cycle cannot be closed. Constraints (4.28) break the cycle because if the three edges are selected the constraint is violated. Moreover, they impose that if edge e_{ab} is selected, no more than one edge from the set $\Delta_{k_1}(a,b,c)\setminus\{e_{ab}\}\cup\{e_{bc}, e_{ca}\}$ can belong to the solution. □

4.4 The Greedy Algorithm

In this section we introduce a greedy algorithm for the Rainbow Spanning Forest Problem. Moreover, to further improve the results we embed it in a multi-start scheme. As described below, the greedy algorithm uses three different selection criteria in order to perform at each iteration the most promising choice. Given a rainbow spanning forest $F = (V, E_F, L_F)$, we denote by $\hat{E}_F$ the set of *feasible edges* in $E \setminus E_F$, that is the set of edges whose endpoints belong to different components of F and such that an edge and the two components to which the endpoints belong to, do not have colors in common. Formally, $\hat{E}_F = \{(u,v) \in E \setminus E_F : u \in T_i,\ v \in T_j,\ T_i \neq T_j,\ \{\ell(T_i) \cap \ell(T_j)\} = \emptyset,\ \{\ell(u,v)\} \cap \{(\ell(T_i) \cup \ell(T_j))\} = \emptyset\}$, where T_i and

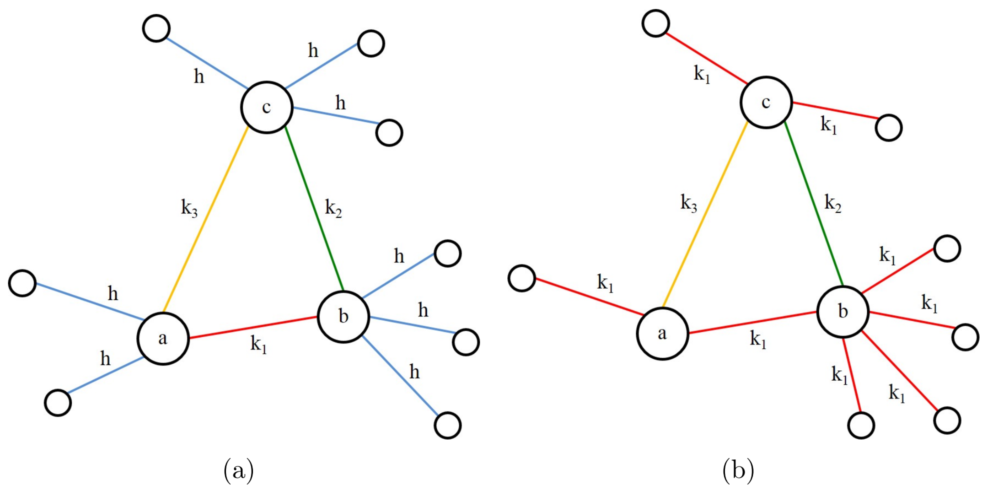


Figure 4.5: (a) An example of valid inequalities 4.27, and, (b) an example of valid inequalities 4.28.

T_j are two generic components of F.
The greedy algorithm starts with the trivial feasible solution $F_0 = (V, E_{F_0}, L_{F_0})$, with $E_{F_0} = L_{F_0} = \emptyset$, in which each vertex is an acyclic and rainbow component. At iteration k the algorithm selects a feasible edge $(u, v) \in \hat{E}_{F_k}$, with $u \in T_i$ and $v \in T_j$, and builds the new rainbow spanning forest F_{k+1} by adding the edge (u, v) in F_k, that is $E_{k+1} = E_k \cup \{(u, v)\}$ and $L_{k+1} = L_k \cup \{\ell(u, v)\}$. It is easy to see that the solution F_{k+1} contains a new larger tree, with respect to F_k, obtained by joining T_i and T_j through (u, v). For this reason, at each iteration the number of trees in the rainbow spanning forest decreases by one, while the number of edges increases by one. The set of feasible edges $\hat{E}_{F_{k+1}}$ is obtained by removing from $\hat{E}_{F_k}$ all edges that are no longer feasible due to the insertion of (u, v). The algorithm stops when there are no more feasible edges. Procedure 3 shows the pseudocode of the greedy algorithm. A key point of the greedy algorithm is the selection of the feasible edge at each iteration. To this end, we define the function $w : \hat{E}_{F_k} \to R$ that associates a weight to the feasible edges in $\hat{E}_{F_k}$ (line 4). The weight $w(u, v)$ represents an estimate of the number of potential improvements that we lose by adding edge (u, v) at iteration k. For this reason, the algorithm always selects the edge of lowest weight (line 5). The details regarding the computation of the weights are given below. Suppose that the graph depicted in Figure 4.6 is the rainbow spanning forest F_k built by the greedy algorithm at iteration k. The dashed edges

Algorithm 3: Greedy Algorithm

```
Input: graph $G = (V, E, L)$.
Output: a rainbow spanning forest $F$ of $G$.
1  Set the number of components $z$ equal to $n$ and $k = 0$
2  $F_k$ trivial starting solution, $\hat{E}_{F_k} = E$
3  while $\hat{E}_{F_k}$ is not empty do
4      update the weights of feasible edges $e \in \hat{E}_{F_k}$
5      select feasible edge $(u, v) \in \hat{E}_{F_k}$ with the min weight
6      $F_{k+1} \leftarrow F_k \cup \{(u, v)\}$
7      update the set of feasible edges $\hat{E}_{F_{k+1}}$
8      update the number of components $z = z - 1$
9      $k = k + 1$
10 return $F_k$.
```

are the feasible edges connecting the four rainbow trees T_1, T_2, T_3, T_4. Let $\delta_k(T_i)$ be the sets of feasible edges incident to the vertices belonging to T_i. Moreover, let $N_k(T_i)$ be the set of trees that can be connected to T_i in F_k through at least one feasible edge. Formally, $\delta_k(T_i) = \{(u, v) \in \hat{E}_{F_k} : u \in V_{T_i} \text{ or } v \in V_{T_i}\}$ and $N_k(T_i) = \{T_j \in T_{F_k} : \exists\, (u, v) \in \hat{E}_{F_k}, u \in T_i, v \in T_j\}$. For instance, in Figure 4.6 we have $\delta_k(T_1) = \{(v_1, v_5), (v_2, v_4), (v_2, v_8), (v_3, v_8), (v_3, v_{11})\}$ and $N_k(T_1) = \{T_2, T_3, T_4\}$. The first information that we want to compute, for each tree T_i, is the "potential" number of trees in T_{F_k} that can be joined to T_i through feasible edges. We call this number the *joining number* and we denote it by M_{T_i}. It is easy to see that $|N_k(T_i)|$ is a trivial upper bound to the value of M_{T_i}. We compute M_{T_i} by solving the maximum matching problem on a bipartite graph B_i in which we define the two sets of vertices $V^1_{B_i}$ and $V^2_{B_i}$ as follows:

- for each $l \in \ell(\delta_k(T_i))$ we define vertex l i.e. $l \in V^1_{B_i}$;
- for each $T_j \in N_k(T_i)$ we define vertex T_j i.e. $T_j \in V^2_{B_i}$.

Moreover, for each feasible edge $(u, v) \in \delta_k(T_i)$, with $v \in T_j$, a corresponding edge in B_i between vertices $\ell(u, v)$ and T_j is introduced. Figure 4.7(a) depicts the bipartite graph B_1 associated to the tree T_1 of the Figure 4.6. Since $\ell(\delta_k(T_1)) = \{6, 8\}$ and $N_k(T_i) = \{T_2, T_3, T_4\}$ then the set of vertices $V^1_{B_1}$ contains only two vertices,

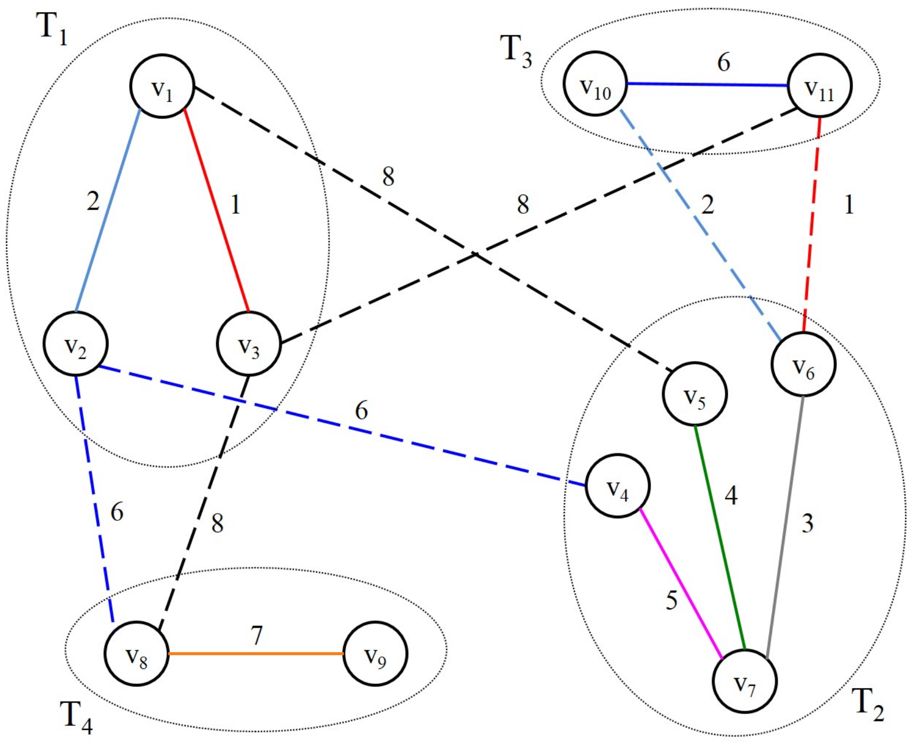


Figure 4.6: Feasible rainbow spanning forest F_k with four components

$V^1_{B_1} = \{6, 8\}$, while the second set is composed of three vertices, $V^2_{B_1} = \{T_2, T_3, T_4\}$. Regarding the edges in B_i, since the edges with color 6 in $\delta_k(T_1)$ connect T_1 to T_2 and to T_4 then in the bipartite graph we connect node 6 with the vertices T_2 and T_4, and so on. The maximum matching of B_1 is equal to two (edges in bold) and then $M_{T_1} = 2$ that is tighter than $|N_k(T_1)| = 3$. This means that T_1 can be joined with, at most, two trees in T_{F_k}. The edges of the maximum matching "suggest" which are the feasible edges to select. For instance, edge $(8, T_4)$ of the maximum matching corresponds to the feasible edge (v_3, v_8) in $\hat{E}_{F_k}$ and by joining the trees T_1 and T_4 through (v_3, v_8), we obtain a larger tree, denoted by $T_{1,4}(v_3, v_8)$, and the new solution F_{k+1} shown in Figure 4.7(b). From now on, given a feasible edge (u, v), with $u \in T_i$ and $v \in T_j$, we denote by $T_{i,j}(u, v)$ the tree obtained by joining T_i and T_j through edge (u, v). Note that in F_{k+1} it is possible to join the new tree $T_{1,4}(v_3, v_8)$ with T_2, through (v_2, v_4), performing, in this way, the two joinings estimated by M_{T_1}. However, the number of joinings carried out does not always

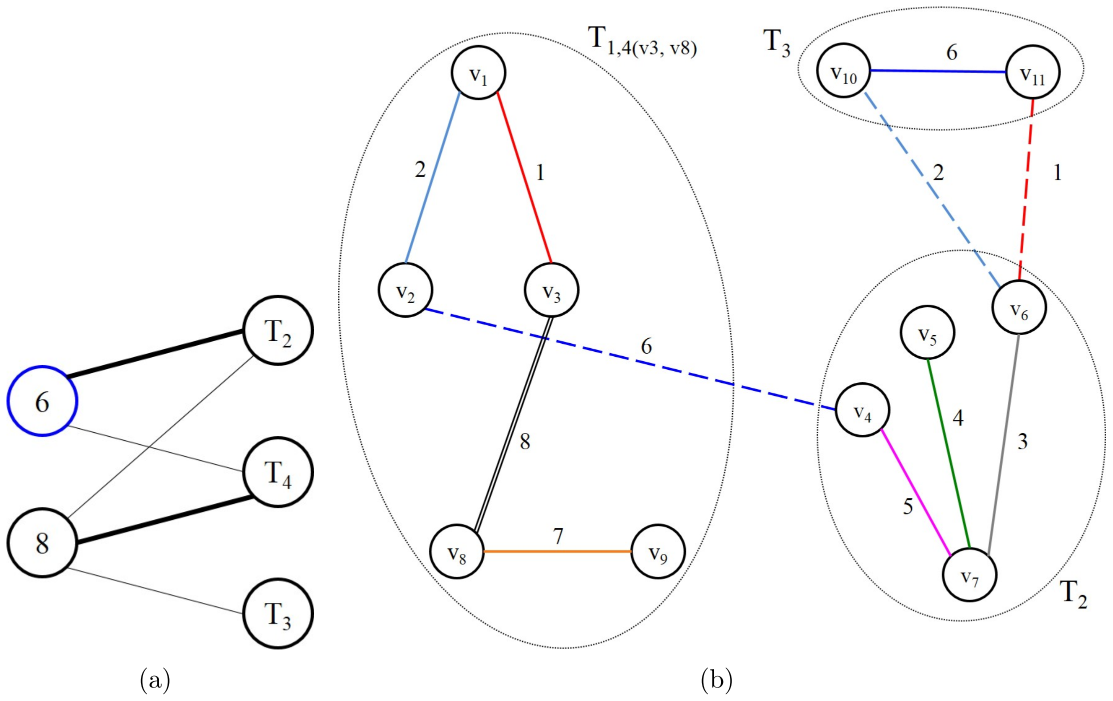


Figure 4.7: Feasible solution F_{k+1} obtained by adding edge (v_3, v_8).

coincide with M_{T_i} because the choice of the feasible edge to select affects the final result and when there are more maximum matchings in B_i there are also more choices available. For instance, Figure 4.8(a) shows another maximum matching of B_1. By joining T_1 and T_3 in F_k through the feasible edge (v_3, v_{11}), as shown in Figure 4.8(b), we obtain the tree $T_{1,3}(v_3, v_{11})$ with $\delta_{k+1}(T_{1,3}(v_3, v_{11})) = \emptyset$. In this case we carried out a single joining compared to the two joinings estimated. The previous example shows that it is necessary to select the feasible edges in an accurate way. For this reason, we propose to first join the trees with lowest *joining number* (*first criterion*). Indeed, there is a high probability that these trees will remain isolated components if they are not joined as soon as possible because a low value of *joining number* means less joining opportunities. For instance, in the solution F_k, depicted in Figure 4.6, $M_{T_1} = M_{T_2} = M_{T_3} = 2$ and $M_{T_4} = 1$ hence T_4 is the tree on which carry out the joining operation with one of its neighbor, T_1 in this case. Since this joining operation can be carried out through edges (v_2, v_8) or (v_3, v_8), then we add another criterion to select the most promising edge. For each edge $(u, v) \in \delta_k(T_4)$, the greedy algorithm computes the *joining number* of the new

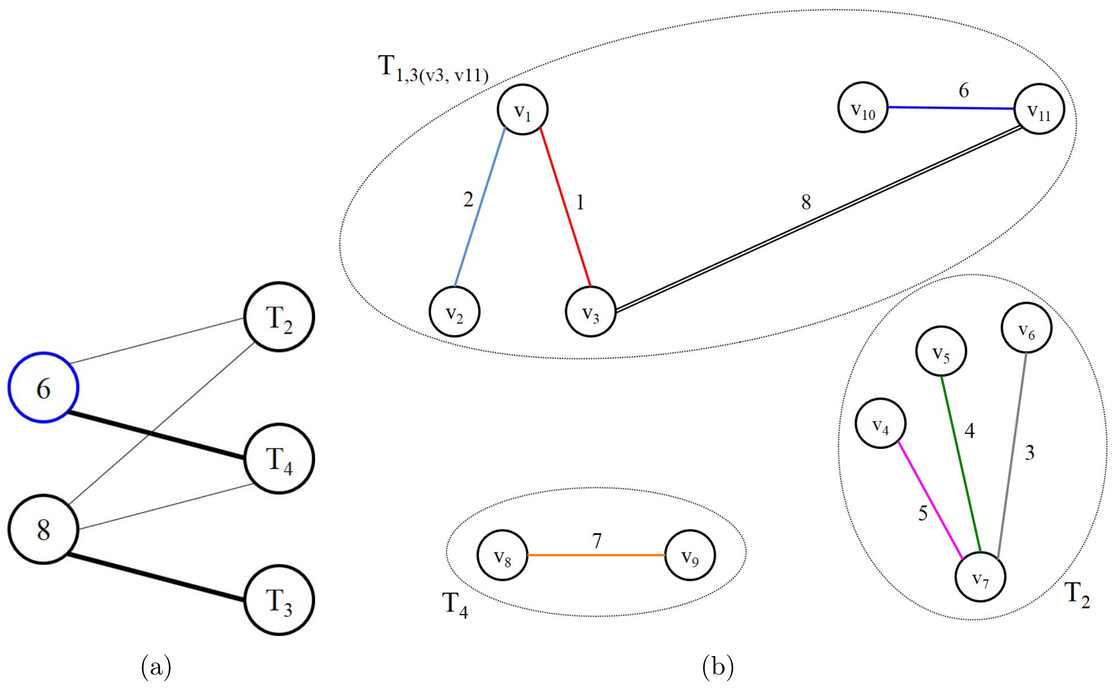


Figure 4.8: Feasible solution F_{k+1} obtained by adding edge (v_3, v_{11}).

tree $T_{1,4}(u,v)$, obtained by joining T_1 and T_4 through (u,v), and it selects the edge (u,v) which value $M_{T_{1,4}(u,v)}$ is the maximum one (*second criterion*).
As previously shown, by selecting edge (v_3, v_8) we obtain the new solution reported in Figure 4.7 with $M_{T_{1,4}(v_3,v_8)} = 1$. Since also $M_{T_{1,4}(v_2,v_8)} = 1$ then we provide a further selection criterion consisting of choosing the edge which color is less frequent in the current solution (*third criterion*). In our example, $\ell(v_3, v_8) = 8$ and there are no edges in E_{F_k} with color 8 while the color 6 of edge (v_2, v_8) belongs to $\ell(E_{F_k})$ due to edge (v_{10}, v_{11}). For this reason, the greedy algorithm selects edge (v_3, v_8) and, thanks to this choice, it finds the optimal solution.

Summarizing, to each feasible edge (u,v) of $\hat{E}_k$, with $u \in T_i$ and $v \in T_j$, we associate a weight according to the following formula:

$$w(u,v) = n \times \min(M_{T_i}, M_{T_j}) - M_{T_{i,j}(u,v)} + \frac{|E_{F_k}(\ell(u,v))|}{z}, \tag{4.29}$$

where $E_{F_k}(c) = \{e \in E_{F_k} : \ell(e) = c\}$. The factor $\min(M_{T_i}, M_{T_j})$ represents the first selection criterion and is multiplied by the coefficient n to ensure that the priority is given to the trees with lowest *joining number*. The term $M_{T_{i,j}(u,v)}$ represents the second selection criterion and since we prefer higher values, this value is subtracted from the first term. Finally, the last term represents the average number of the color $\ell(u,v)$ in F_k. The weights for the feasible edges of the solution in Figure 4.6 are the following:

$$\begin{aligned}
w(v_1, v_5) &= 10 \times \min(2,2) - 1 + 0 = 19 \\
w(v_2, v_4) &= 10 \times \min(2,2) - 1 + \frac{1}{4} = 19.25 \\
w(v_2, v_8) &= 10 \times \min(2,1) - 1 + \frac{1}{4} = 9.25 \\
w(v_3, v_8) &= 10 \times \min(2,1) - 1 + 0 = 9 \\
w(v_3, v_{11}) &= 10 \times \min(2,2) - 0 + 0 = 20 \\
w(v_6, v_{10}) &= 10 \times \min(2,2) - 0 + \frac{1}{4} = 20.25 \\
w(v_6, v_{11}) &= 10 \times \min(2,2) - 0 + \frac{1}{4} = 20.25
\end{aligned}$$

4.4.1 The multi-start scheme

The greedy algorithm starts from the trivial solution F_0 and, at each step k, adds a new edge from the set of feasible edges $\hat{E}_{F_k}$ until $\hat{E}_{F_k}$ is not empty. The selection of edges is carried out according to their weights, that are dynamically updated according to the choices performed in the previous steps. This means that a bad choice carried out in the first steps can heavily affect the quality of the final solution produced. We face this problem by embedding the greedy algorithm in a multi-start scheme which, by performing a deeper exploration of the solution space, finds better solutions. More specifically, the multi-start algorithm invokes the greedy algorithm several times with different starting feasible solutions in which a *first edge* is fixed. This first edge is chosen from a set of edges $\tilde{E}$ that the multi-start algorithm builds at the beginning of the computation. The key point is how the set $\tilde{E}$ is built because i) the effectiveness of the multi-start algorithm depends on the edges in this set, and ii) the performance of the multi-start algorithm depends on the cardinality of $\tilde{E}$, since the greedy algorithm is invoked $|\tilde{E}|$ times. To build $\tilde{E}$, the multi-start algorithm first computes the weight of all edges, according to equation (4.29), and later, for each vertex $v \in V$, it inserts in $\tilde{E}$ the edge of $\delta(v)$ with the lowest weight. If this edge already belongs to $\tilde{E}$ it is ignored and a new vertex is selected. This construction ensures that the greedy algorithm will build each solution starting from a different vertex. The selection of the most promising edge for each vertex rather than the most promising at all, i.e. the edges with n lowest weights, allows a better exploration of the solution space. Since the cardinality of $\tilde{E}$ affects the performance of the multi-start algorithm, we have to control the growth of this cardinality as the instances size increases. This choice, on the one hand, preserves the performance of the algorithm even on the larger instances and, on the other hand, reduces the quality of solutions found on these instances. However, this reduction does not have a major impact since fixing a single edge at beginning of the computation does not significantly affect the final solution when the instance size increases. For this reason, we bound the cardinality of $\tilde{E}$ as follows:

$$|\tilde{E}| \leq \min\left\{ \left\lfloor 10 + \frac{500}{\sqrt{n}} \right\rfloor, n \right\}. \tag{4.30}$$

Finally, the multi-start algorithm invokes $|\tilde{E}|$ times the greedy algorithm and returns the best solution among the $|\tilde{E}|$ solutions identified.

4.5 Computational results

In this section we present computational results obtained by solving the ILP1, the greedy algorithm and the multi-start method. The greedy algorithm and the multi-start scheme were coded in Java and computational experiments were performed on a 64-bit GNU/Linux operating system, 96 GB of RAM and one processor Intel Xeon X5675 running at 3.07 GHz. The mathematical model ILP1 was coded in C and solved using IBM ILOG CPLEX 12.5 on the same machine. To the best of our knowledge, there are no available benchmark instances for the RSFP and we therefore used randomly generated instances. Some instances considered in the current study were introduced by Silvestri et al. [51] for the Rainbow Cycle Cover Problem. Here we create additional instances according to the procedure used in [51], and described next.

Each instance is characterized by the number of vertices n (size), the number of edges m and the number of colors l. Given the number of vertices n, the number of edges is set to $m = \lceil \frac{n(n-1)}{2} \times d + n \rceil$, with $d \in \{0.1, 0.2, 0.3\}$, and the number of colors is set to $\lceil \frac{1}{2} \log(m) \rceil$, $\lceil \log(m) \rceil$ and $\lceil 2 \log(m) \rceil$. Note that the number of colors is always less than n, therefore the optimal solution cannot be a tree and hence the instances are not polynomially solvable. The total number of different scenarios is nine for each size. Each scenario is composed by five different instances having the same number of vertices, edges and colors and the results reported in each line of the tables are the average values computed over these five instances. The combination of all these parameters allows us to verify how the effectiveness and performance of our algorithms are affected by the number of vertices, the density of the graph and the number of colors. The scenarios are divided into two groups: the small scenarios, where the value of n ranges from 20 to 50 with a step equal to 10, and the large scenarios, where the value of n ranges from 100 to 400 with a step equal to 100.

Note that the density of the generated instances is low (up to 0.3) in order to obtain meaningful results. It is easy to see that for higher density values the value of the optimal solution almost always coincides with the trivial lower bound $LB = \lceil \frac{n}{l+1} \rceil$. To explain this we need to introduce the definition of *rainbow star.*

Definition 4.2. *Given an edge-colored tree $T = (V_T, E_T, L_T)$ with $|V_T| = k + 1$, this tree is a rainbow star if and only if*

- *each edge has a different color, i.e. $|L_T| = |E_T| = k$;*
- *k vertices are leaves, i.e. k vertices have degree equal to one in T, and one vertex has degree k.*

Note that, for any vertex $v \in V$, as the density increases the probability that $\delta(v)$ contains a rainbow star increases. Therefore, for higher density is high the probability of obtaining a trivial optimal solution with $\lceil \frac{n}{l+1} \rceil$ rainbow stars.

Table 4.1 reports the results of the mathematical model (ILP1), of the Greedy (GA) and of the multi-start (MS) algorithms, on small scenarios. The first four columns report the characteristics of each scenario: scenario ID, the number of vertices (n), the number of edges (m), the number of colors (l), respectively. The

ID	n	m	l	ILP1		GA		MS		GAP	
				Value	Seconds	Value	Seconds	Value	Seconds	GA	MS
1	20	39	3	6.6	2.94	8.0	0.01	7.0	0.10	1.4	0.4
2			6	4.2	3.74	5.0	0.02	4.8	0.10	0.8	0.6
3			11	2.2	1.10	3.0	0.02	2.8	0.13	0.8	0.6
4	20	58	3	5.6	3.90	6.4	0.02	6.4	0.13	0.8	0.8
5			6	3.2	2.14	4.4	0.03	3.6	0.15	1.2	0.4
6			12	2.0	1.21	2.2	0.04	2.0	0.05	0.2	**0.0**
7	20	77	4	4.0	2.30	4.8	0.03	4.4	0.11	0.8	0.4
8			7	3.0	1.63	3.2	0.05	3.0	0.07	0.2	**0.0**
9			13	2.0	1.63	2.0	0.06	2.0	0.06	**0.0**	**0.0**
10	30	74	4	8.4	129.42	8.8	0.03	8.8	0.20	0.4	0.4
11			7	5.0	204.63	6.2	0.03	5.4	0.25	1.2	0.4
12			13	3.0	8.02	3.6	0.06	3.4	0.16	0.6	0.4
13	30	117	4	6.8$^{(1)}$	2199.73	8.4	0.04	7.4	0.33	1.6	0.6
14			7	4.2	16.72	4.8	0.07	4.4	0.27	0.6	0.2
15			14	2.6	51.51	3.2	0.09	3.0	0.49	0.6	0.4
16	30	161	4	6.4	36.88	8.0	0.07	6.4	0.32	1.6	**0.0**
17			8	4.0	17.27	4.6	0.09	4.0	0.12	0.6	**0.0**
18			15	2.0	18.84	3.0	0.12	2.2	0.39	1.0	0.2
19	40	118	4	10.2	2283.43	12.6	0.03	12.0	0.29	2.4	1.8
20			7	6.4$^{(2)}$	4711.31	8.2	0.06	7.4	0.41	1.8	1.0
21			14	3.4$^{(2)}$	4348.36	5.0	0.08	4.0	0.51	1.6	0.6
22	40	196	4	8.8	2336.74	11.0	0.08	9.8	0.48	2.2	1.0
23			8	5.0	121.37	6.2	0.10	5.2	0.43	1.2	0.2
24			16	3.0	82.75	3.8	0.14	3.2	0.33	0.8	0.2
25	40	274	5	7.0	89.99	7.8	0.11	7.0	0.22	0.8	**0.0**
26			9	4.0	86.37	5.0	0.14	5.0	1.00	1.0	1.0
27			17	3.0	139.92	3.2	0.18	3.0	0.21	0.2	**0.0**
28	50	173	4	12.6$^{(4)}$	8994.51	16.0	0.06	14.6	0.35	3.4	2.0
29			8	9.0$^{(4)}$	10652.42	10.4	0.08	9.2	0.46	1.4	0.2
30			15	4.8$^{(3)}$	6536.58	5.4	0.12	5.4	0.54	0.6	0.6
31	50	295	5	9.0	618.57	11.0	0.10	9.8	0.55	2.0	0.8
32			9	5.2$^{(1)}$	2318.95	6.6	0.14	6.0	0.84	1.4	0.8
33			17	3.0	1009.18	4.4	0.17	4.0	1.11	1.4	1.0
34	50	418	5	9.0	294.84	9.4	0.14	9.0	0.59	0.4	**0.0**
35			9	5.0	295.96	6.0	0.20	6.0	1.19	1.0	1.0
36			18	3.0	337.71	4.0	0.28	3.2	0.67	1.0	0.2

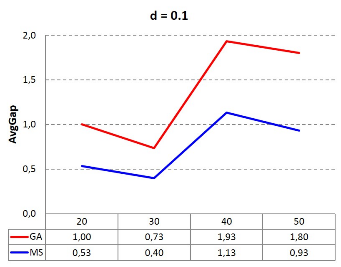
Table 4.1: Test results of ILP1 model, GA and MS algorithms on the small scenarios.

columns ILP1, GA and MS are divided into two subcolumns (Value and Seconds) reporting the solution value and the computing time in seconds, respectively. We have imposed a time limit of $10,800$ seconds. Whenever α instances of a scenario were not solved to optimality by ILP1, within the time limit, we report (α) close to the solution value, therefore the value reported is an upper bound on the optimal solution value. In this cases we say that a ILP1 *failure* occurs and we refer to the solutions with the symbol (α) as the *best bound* solutions. The last column (GAP) reports the gaps between the solution computed by ILP1 and the solutions found by GA and by MS, respectively. We mark in bold the gaps equal to zero to highlights the scenarios where the algorithms find an optimal solution.

The results of Table 4.1 show that ILP1 finds an optimal solution on 29 out of 36 scenarios and that the hardest scenarios to solve are those with lowest density ($d = 0.1$). Indeed, five of the seven ILP1 failures occur on scenarios with density 0.1 (ID n° 20, 21, 28, 29 and 30). The remaining two failures occur on scenarios with density 0.2 (ID n° 13 and 32). Finally, on the scenarios with density 0.3 ILP1 always finds the optimal solution and the CPU time is always lower than six minutes. These results highlight the good performance of our mathematical model that almost always finds the optimal solution for the instances with a density greater than or equal to 0.2. The worst results are obtained on the scenarios with density 0.1 which from now on we denote as the *critical scenarios*. This is probably due to the low number of optimal solutions available. Indeed, as the density decreases, the number of equivalent feasible solutions decreases. Therefore there may be a really small number of optimal solutions, and then it results harder to find one of them.

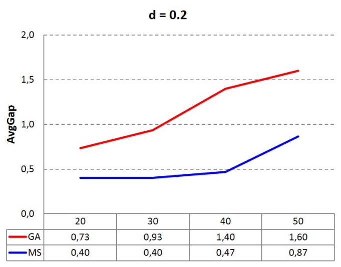
It is interesting to observe that for the greedy and the multi-start algorithms the critical scenarios are also the hardest to solve. Indeed, the highest gap values occur on these scenarios and on the critical scenario n° 28, the *peak* (the maximum gap value) of GA and MS occurs. This peak is equal to 3.4 for GA and 2.0 for MS.

Regarding the effectiveness of the two algorithms, as expected the results of MS are much better than those of GA. MS finds the optimal solution on eight out of 29 scenarios where the optimal solution is known while GA finds the optimal solution only one time (n° 9). Moreover, the gap value of MS is lower than or equal to one on 34 out of 36 scenarios while for GA this condition holds only 21 times.



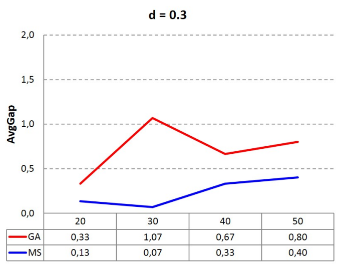
	20	30	40	50
GA	1,00	0,73	1,93	1,80
MS	0,53	0,40	1,13	0,93

(a)



	20	30	40	50
GA	0,73	0,93	1,40	1,60
MS	0,40	0,40	0,47	0,87

(b)



	20	30	40	50
GA	0,33	1,07	0,67	0,80
MS	0,13	0,07	0,33	0,40

(c)

Figure 4.9: *The AvgGap of GA and MS algorithms on the larger scenarios.*

From the gap values it is evident that the density of the graph is the main parameter affecting the effectiveness of GA and MS. In order to analyze the trend of these two algorithms on the scenarios with the same density but with an increasing number of vertices, we introduce another measure, the *AvgGap*. This measure represents the average gap values computed on the scenarios with the same number of vertices and edges. For instance, for $n = 20$ and $m = 39$, the AvgGap of MS is equal to $(0.4 + 0.6 + 0.6)/3 = 0.53$.

In Figure 4.9 the AvgGap of GA (in red) and MS (in blue) are plotted for the small scenarios with $d = 0.1$, $d = 0.2$ and $d = 0.3$, respectively. Here on the x-axis and y-axis the number of vertices and the AvgGap are reported, respectively.

The graphic in Figure 4.9 shows that the two algorithms have a similar growth but the AvgGaps of MS are nearly half of the AvgGaps of GA. Again, the highest values for the AvgGap are obtained when $d = 0.1$ (Figure 4.9(a)). Despite that, the AvgGap of GA is always lower than two while the AvgGap of MS is always lower than 1.2. On the scenarios with $d = 0.2$ (Figure 4.9(b)) the growth is more regular for both the algorithms, even if it is slower for MS. Indeed, the AvgGap of MS ranges from 0.40 for $n = 20$ to 0.87 for $n = 50$ while the AvgGap of GA ranges from 0.73 for $n = 20$ to 1.60 for $n = 50$. It is interesting to observe that even on these small scenarios the AvgGap between the two algorithm can be significantly different. For instance, for $n = 40$ the AvgGap of GA is three times greater than the AvgGap of MS. Finally, for $d = 0.3$ (Figure 4.9(c)) the results of both algorithms improve, with a maximum AvgGap equal to 1.07 for GA and 0.40 for MS. Even in this case, the AvgGap of MS is always the half of AvgGap of GA and, on the scenarios with $n = 30$, the difference grows up to one. By observing the three graphics of Figure 4.9 it is evident that our algorithms are more effective when the density grows.

Regarding the performance, both the algorithms are very fast with a negligible running time because it is rarely greater than one second.

Table 4.2 shows the results obtained by GA and MS algorithms on the large scenarios. Since the solutions of the ILP1 are not available for these scenarios, we compare the solution values produced by GA and MS and we compute the GAP on these values.

Despite the negligible computational times on the small scenarios, here the performance of the two algorithms are much different. In particular, GA is one order of magnitude faster than MS, often its running time is lower than one minute and, in the worst case (n° 71) it requires 77 seconds. More computational time is required by MS that in the worst case (n° 71) spends 857 seconds (n° 71). Anyway, most of the scenarios are solved by MS in less than five minutes. It is clear that the significant performance difference on the large scenarios is due to the fact that MS invokes several times the GA algorithm. The trend of the running times for both algorithms shows that the performance are mainly affected by the density of the scenarios. As the density increases as the computation time grows.

If, on the one hand, the MS algorithm is slower than GA, on the other hand it results more effective as shown in the GAP column. Indeed, on all the scenarios MS finds solution always better than GA. In particular, the GAP value is greater than or equal to one on 20 out of 36 scenarios and in six cases this GAP is greater than two. We observe also a GAP equal to 21.4 on the scenario n° 67 where the solution of GA is very poor.

It is interesting to observe that the lowest GAP values occur on the scenarios with $d = 0.3$ on which the two algorithms require more computational time. Our conjecture is that, since these scenarios contains more edges, there are more local minimums with a good solution value. As a consequence, when GA is trapped into a local minimum the value found is good and the GAP from MS low. Instead, the situation changes on the scenarios with fewer edges ($d = 0.1$ and $d = 0.2$) where the number of good local minimums is lower. In this case, there are more chances that GA is trapped into a poor local minimum producing a solution value far from the solution value of MS.

ID	***n***	***m***	***l***	**GA**		**MS**		**GAP**
				Value	Seconds	Value	Seconds	
37	100	595	5	26.6	0.18	24.8	0.73	1.8
38			10	12.8	0.25	11.8	1.15	1.0
39			19	8.2	0.33	7.2	1.45	1.0
40	100	1090	6	17.0	0.39	15.6	1.52	1.4
41			11	10.6	0.54	10.2	2.00	0.4
42			21	7.2	0.57	6.2	2.64	1.0
43	100	1585	6	15.8	0.71	15.2	1.27	0.6
44			11	10.0	0.92	9.2	2.11	0.8
45			22	6.0	0.96	5.6	4.50	0.4
46	200	2190	6	46.0	0.77	43.6	3.47	2.4
47			12	20.0	1.09	18.6	5.05	1.4
48			23	13.8	1.39	13.2	7.62	0.6
49	200	4180	7	30.0	2.46	27.0	12.63	3.0
50			13	17.6	2.17	16.6	16.96	1.0
51			25	11.8	2.57	10.4	23.29	1.4
52	200	6170	7	26.2	2.77	26.2	25.26	0.0
53			13	16.0	3.71	15.2	25.50	0.8
54			26	9.4	4.50	9.0	45.37	0.4
55	300	4785	7	58.2	1.72	54.8	12.32	3.4
56			13	27.8	2.23	26.4	19.26	1.4
57			25	19.6	2.97	18.0	28.99	1.6
58	300	9270	7	47.0	4.81	41.8	50.44	5.2
59			14	22.2	7.14	22.0	82.13	0.2
60			27	15.0	8.98	14.2	96.14	0.8
61	300	13755	7	39.4	11.58	39.2	125.62	0.2
62			14	21.4	18.83	21.0	203.72	0.4
63			28	12.4	18.63	11.8	177.91	0.6
64	400	8380	7	86.0	4.37	79.2	43.33	6.8
65			14	33.0	6.51	31.8	69.59	1.2
66			27	25.0	8.78	22.8	97.05	2.2
67	400	16360	7	82.0	16.29	60.6	176.52	21.4
68			14	29.0	25.31	28.4	279.07	0.6
69			28	18.4	27.01	17.0	306.68	1.4
70	400	24340	8	46.6	43.30	45.2	247.19	1.4
71			15	26.4	77.16	26.0	857.59	0.4
72			30	15.2	61.59	14.4	718.85	0.8

Table 4.2: Test results of GA and MS on the large scenarios.

Chapter 5

A Branch-and-Cut Algorithm for the Minimum Branch Vertices Spanning Tree Problem

“ *All religions, arts and sciences are branches of the same tree.* ”

Albert Einstein

5.1 Introduction

In Chapter 2 we made a brief overview of some problems known in the combinatorial optimization literature, concerning spanning tree and cycle cover. The *Minimum Branch Vertices Problem* belongs to the class of $\mathcal{NP}$-hard spanning tree problems. A slightly different version of this chapter was submitted to Computers & Operations Research.

Let $G = (V, E)$ be a connected undirected graph, with $n = |V|$ vertices and $m = |E|$ edges, the Minimum Branch Vertices Problem (MBVP) looks for a spanning tree T of G with the minimum number of branch vertices, i.e. vertices having a degree greater than or equal to three. For the input graph given on the

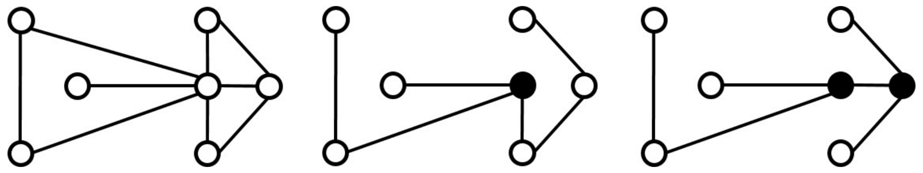


Figure 5.1: For a given graph on the left, two spanning trees with one and two branch vertices.

left of Figure 5.1, we depict two spanning trees with different numbers of branch vertices. The spanning tree in the middle has one branch vertex and the one on the right has two. There is no spanning tree without branch vertices, therefore the one in the middle is the optimal solution.

The MBVP finds application in the context of optical networks. In such networks, the aim is to send an optical signal from a source node to all the nodes of the network. The optical signal has to be split whenever it enters a node having degree greater than two and to split the signal is necessary to locate an appropriate network switch at all the branch vertices. These switches can significantly increase the cost of the network.

The MBVP was introduced by Gargano et al. [22], who proved it to be $\mathcal{NP}$-hard. Since then, the problem has been extensively investigated by several authors [6], [7], [10], [37], [39], [49], [50], [52]. Carrabs et al. [6] consider four IP formulations. The first formulation contains the subtour elimination constraints introduced by Dantzig et al. [15]. Due to the exponential number of constraints, the authors consider this formulation not suitable to be tested on instances of significant size, but they solve it in a Lagrangian relaxation fashion. The second formulation is the most studied. It guarantees connectivity by sending from a source vertex one unit of flow to every other vertex of the graph. The third formulation is based on a multi-commodity flow. The fourth formulation makes use of the Miller-Tucker-Zemlin subtour elimination constraints [40]. Finally, Marín [37] presents a branch-and-cut algorithm based on a strengthened single commodity flow formulation. The author also provides a two-stage heuristic to reduce the computational time and to produce good feasible solutions when the optimum cannot be found within a reasonable time.

The purpose of this chapter is to develop new integer linear formulations and a polyhedral-based exact branch-and-cut algorithm for the MBVP. The remainder of the chapter is organized as follows. In Section 5.2, we formulate the problem as an integer linear program with variables associated with the edges of G. In this section, we also investigate some properties of the problem. In Section 5.3, the dimension of the polyhedron is derived as well as some facet related results, and some valid inequalities are introduced. In Section 5.4, we present a directed graph reformulation and we adapt to this formulation some properties of the problem and some valid inequalities to yield a hybrid formulation. The branch-and-cut algorithm is described in Section 5.5. Comparative computational results are presented in Section 5.6.

5.2 Undirected formulation, properties and bounds

The MBVP can be formulated as an integer linear program (ILP) with undirected variables as follows. Let x_e be a binary variable equal to 1 if and only if edge $e \in E$ belongs to the spanning tree T. For the sake of simplicity, in the following we will refer to variables x_e, which are variables associated to the edges of the undirected graph G, as *undirected variables*. Moreover, let y_v be a binary variable equal to 1 if and only if vertex $v \in V$ is a branch vertex, i.e. v has degree greater than or equal to 3 in T. In addition, for $S \subset V$, define $E(S) = \{e = (v,u) \in E : v, u \in S\}$ and $\delta(S) = \{e = (v,u) \in E : v \in S,\ u \in V \setminus S\}$. If $S = \{v\}$, we simply write $\delta(v)$ instead of $\delta(\{v\})$. We write $\delta_{G'}(v)$ to denote the set of edges incident to v in a subgraph G' of G. The ILP formulation is then

$$\text{minimize } z = \sum_{v \in V} y_v \tag{5.1}$$

subject to

$$\sum_{e \in E(S)} x_e \leq |S| - 1 \qquad S \subset V,\ |S| \geq 3 \tag{5.2}$$

$$\sum_{e \in E} x_e = n - 1 \tag{5.3}$$

$$\sum_{e \in \delta(v)} x_e - 2 \leq (|\delta(v)| - 2) y_v \qquad v \in V \tag{5.4}$$

$$x_e \in \{0,1\} \qquad e \in E \tag{5.5}$$

$$y_v \in \{0,1\} \qquad v \in V. \tag{5.6}$$

In this formulation, constraints (5.2) are the well-known Dantzig, Fulkerson and Johnson [15] subtour elimination constraints. They guarantee that the optimal solution contains no cycles. Constraint (5.3) forces the selection of exactly $n-1$ edges, which together with constraints (5.2) ensure that the optimal solution is a tree. Constraints (5.4) are logical constraints linking the binary variables x_e with the binary variables y_v. They guarantee that a vertex v is branch whenever at least three edges incident to it are selected. Even if they do not explicitly force $y_v = 0$ when $\sum_{e \in \delta(v)} x_e \leq 2$ holds, due to the objective function used, this condition is satisfied for any optimal solution. The objective function (5.1) requires the minimization of the number of branch vertices. This formulation is already known (see Melo et. al [38]) and represents a stronger version of one of the mathematical formulations proposed by Carrabs et al. [6]. Note that in order to make variables y_v represent exactly a set of branch vertices, we need the additional constraints

$$2y_v \leq \sum_{e \in \delta(v)} x_e - 1 \qquad v \in V. \tag{5.7}$$

Constraints (5.7) together with constraints (5.4) guarantee that y_v, $v \in V$, is equal to 1 if and only if vertex v is branch. Let P_L denotes the polytope obtained by the linear relaxation of (5.5) and (5.6), plus constraints (5.2)−(5.4) and (5.7).

5.2.1 Spanning tree properties

This subsection is devoted to the description of some properties that a spanning tree must satisfy. We will also do some observations that will allow us to preprocess the instances. For a given vertex v, we can write the set of incident edges $\delta(v)$ as

$$\delta(v) = \delta^L(v) \cup \delta^I(v), \tag{5.8}$$

where $\delta^L(v) = \{(v,u) \in \delta(v) : |\delta(u)| = 1\}$ and $\delta^I(v) = \{(v,u) \in \delta(v) : |\delta(u)| > 1\}$. As Marín [37] observed, each edge belonging to the set $\delta^L(v)$, for a given vertex v, must belong to an optimal tree T:

$$x_e = 1 \qquad\qquad v \in V, \quad e \in \delta^L(v). \tag{5.9}$$

Moreover,

$$y_v = 0 \qquad\qquad v \in V : \ |\delta(v)| \leq 2 \tag{5.10}$$

$$y_v = 1 \qquad\qquad v \in V : \ |\delta^L(v)| \geq 2. \tag{5.11}$$

Note that for each vertex v such that $|\delta^L(v)| = 1$, constraints (5.4) and (5.7) become respectively

$$\sum_{e \in \delta^I(v)} x_e - 1 \leq (|\delta^I(v)| - 1) y_v \qquad\qquad v \in V : \ |\delta^L(v)| = 1 \tag{5.12}$$

$$y_v \leq \sum_{e \in \delta^I(v)} x_e - 1 \qquad\qquad v \in V : \ |\delta^L(v)| = 1. \tag{5.13}$$

To ensure the connectivity property, the inequalities

$$\sum_{e \in \delta^I(v)} x_e \geq 1 \qquad\qquad v \in V : |\delta^I(v)| > 0 \tag{5.14}$$

must be satisfied.
Marín [37] defines a *bridge* as an edge $e \in E$ such that the graph $(V, E \setminus \{e\})$ becomes disconnected and defines 2-*cocycle* a set of two edges $\{e, f\} \subset E$ such that the graph $(V, E \setminus \{e, f\})$ becomes disconnected, but e and f are not bridges. An easy observation is that all bridges of a connected graph must belong to the edge set of any spanning tree:

$$x_e = 1 \qquad\qquad e \in E : \ (V, E \setminus \{e\}) \text{ is disconnected.} \tag{5.15}$$

Moreover, at least one of the edges of a 2-cocycle set must belong to any feasible solution:

$$x_e + x_f \geq 1 \qquad\qquad e, f \in E : \{e, f\} \text{ is a 2-cocycle.} \tag{5.16}$$

Note that all edges belonging to the set $\bigcup_{v \in V} \delta^L(v)$ are particular bridges. Removing any one of them isolates a vertex. To identify the bridges and the 2-cocycle sets Marín [37] uses an algorithm proposed by Schmidt [47].
In graph theory [2] a *cut vertex* is a vertex $v \in V$ such that the graph $G \setminus v = (V \setminus \{v\}, E \setminus \delta(v))$ is disconnected. Let $\bar{c}_v$ be the number of connected components of the graph $G \setminus v$ and let $C_i(v) = (V_{C_i}(v), E_{C_i}(v))$, $i = 1, \dots, \bar{c}_v$, be the corresponding connected components, such that $\bigcup_{i=1}^{\bar{c}_v} V_{C_i}(v) = V \setminus \{v\}$ and $\bigcup_{i=1}^{\bar{c}_v} E_{C_i}(v) = E \setminus \delta(v)$. For a given cut vertex v, we can write the set of incident edges $\delta(v)$ as

$$\delta(v) = \bigcup_{i=1}^{\bar{c}_v} \delta^i(v), \tag{5.17}$$

where $\delta^i(v) = \{(v, u) \in \delta(v) : u \in V_{C_i}(v)\}$. Gargano et al. [22] observed that any cut vertex v such that $G \setminus v$ becomes disconnected in at least three connected components is necessarily a branch vertex in any spanning tree of G. Therefore, if we denote V_B the set of cut vertices, it is easy to see that

$$y_v = 1 \qquad\qquad v \in V_B : \bar{c}_v \geq 3. \tag{5.18}$$

Moreover, it is easy to see that the following inequalities hold true:

$$\sum_{e \in \delta^i(v)} x_e - 1 \leq (|\delta^i(v)| - 1) y_v \qquad\qquad v \in V_B : \bar{c}_v = 2, \ i = 1, 2 \tag{5.19}$$

$$\sum_{e \in \delta^i(v)} x_e \geq 1 \qquad\qquad v \in V_B, \ i = 1, \dots, \bar{c}_v. \tag{5.20}$$

Note that inequalities (5.19) and (5.20) are a restricted version of (5.4) and (5.14) respectively. A connected graph G is *2-connected* if G contains no *cut vertex*. In this work we call *2-disconnected* a connected graph G such that G contains no cut vertex v such that $G \setminus v$ is disconnected into more than two components. For

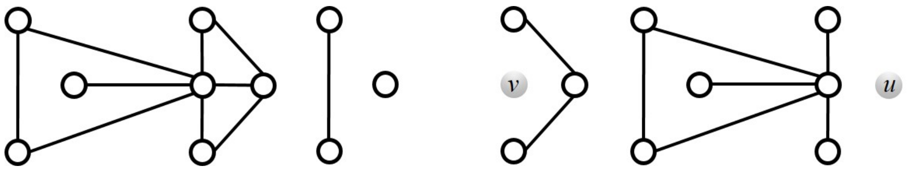


Figure 5.2: For a given graph on the left, v is a cut vertex u is not a cut vertex.

the input graph given on the left of Figure 5.2 we depict two examples. Given the graph on the left, the graph in the middle is obtained by removing the gray vertex v and the set of incident edges $\delta(v)$. The graph $G \setminus v$ results disconnected, therefore vertex v is a cut vertex. Moreover, it is easy to observe that $G \setminus v$ is disconnected into three components, i.e. $\bar{c}_v = 3$, and hence v is branch and the graph G is not 2-disconnected. The graph on the right is obtained by removing the gray vertex u and the set of incident edges $\delta(u)$. $G \setminus u$ is not disconnected, therefore u is not a cut vertex. Note that if all the vertices of a connected graph are cut vertices or have degree equal to one, the graph is a tree. For this reason, in the remainder of this section we assume that G contains at least one cycle.

Lemma 5.1. *Let $G = (V, E)$ be a 2-disconnected graph. Then, for any $v \in V$, there exists a spanning tree T in G such that v is not a branch vertex in T.*

Proof. Since G is 2-disconnected, $G \setminus v$ can be connected or disconnected into two components. If it is connected, there exists a spanning tree T_v in $G \setminus v$, therefore $T = T_v \cup \{e\}$ is a spanning tree in G, for any $e \in \delta(v)$, such that $|\delta_T(v)| = 1$. If $G \setminus v$ is disconnected, there exist two spanning trees T_1 and T_2 in $C_1(v)$ and $C_2(v)$, respectively. Hence, for an arbitrary $e_1 \in \delta^1(v)$ and $e_2 \in \delta^2(v)$, $T = T_1 \cup T_2 \cup \{e_1, e_2\}$ is a spanning tree in G such that $|\delta_T(v)| = 2$. □

5.3 Polyhedral analysis of the undirected formulation

In this section we derive some polyhedral results for the MBVP. We assume that $G = (V, E)$ is a 2-disconnected graph containing cycles. Let P be

$$P = \text{conv}\big\{(x, y) \in \mathbb{R}^{|E|+|V|} : (x, y) \text{ satisfies } (5.2) - (5.6)\big\}. \tag{5.21}$$

Note that we define P as the convex hull of the solutions satisfying (5.2) – (5.6), namely solutions such that y does not necessarily represent exactly a set of branch vertices. The assumptions on G are not a real restriction for studying P. If G is a connected acyclic graph, G is already a tree. Moreover, assume G is not 2-disconnected and let $v \in V$ be a vertex such that $G \setminus v$ becomes disconnected into $s \geq 3$ connected components, this means that v is going to be a branch vertex in any spanning tree of G. Melo et al. [38], propose a *obligatory branches based decomposition* and a *cut edges based decomposition* which, starting from $G = (V, E)$, build a new graph $G_0^{'} = (V_0^{'}, E_0^{'})$ that becomes disconnected if at least one compulsory branch or one cut edge is encountered. Moreover, they provide the following proposition

Proposition 5.1. *An optimal solution to the minimum branch vertices problem can be obtained from the solutions of the s connected components of $G_0^{'}$ and its optimal value is*

$$z = |L_0| + \sum_{i=1}^{s} z_i, \tag{5.22}$$

where L_0 represents the set of compulsory vertices, i.e. vertices that are branch in any spanning tree of G, and z_i, $i = 1, \ldots, s$, the optimal value obtained for the i^{th} connected component.

It is easy to observe that Proposition 5.1 holds true even if only the compulsory branch decomposition is applied and this prove that if the graph is not 2-disconnected, the problem can be decomposed into separate subproblems. Therefore for the rest of this section, we assume G is 2-disconnected. In order to provide our polyhedral results, we need some preliminary results.

Definition 5.1. *A polyhedron $S = \{x \in \mathbb{R}^k : Ax \leq b\}$ is full-dimensional if $dim(S) = k$, where (A, b) is an $h \times (k+1)$ matrix.*

Let $M = \{1, \ldots, h\}$, $M^= = \{i \in M : a^i x = b_i$ for all $x \in S\}$ and $M^{\leq} = \{i \in M : a^i x < b_i$ for some $x \in S\} = M \setminus M^=$, where a^i represents the i^{th} row of A. Let $(A^=, b^=)$ and $(A^{\leq}, b^{\leq})$ the corresponding rows of (A, b). According to this notation, the following theorem holds true (see Chapter 3, Theorem 3.17 of Conforti et al. [12]):

Theorem 5.1. *Let $S = \{x \in \mathbb{R}^k : Ax \leq b\}$ be a nonempty polyhedron. Then*

$$\mathit{aff}(S) = \{x \in \mathbb{R}^k : A^{=}x = b^{=}\} = \{x \in \mathbb{R}^k : A^{=}x \leq b^{=}\}. \tag{5.23}$$

Furthermore, $\mathit{dim}(S) = k - \mathit{rank}(A^{=})$.

We represent subsets of vertices and edges by their characteristic vectors $y \in \mathbb{B}^n$ and $x \in \mathbb{B}^m$, respectively. Therefore, $V' \subseteq V$ is represented by the vector $y^{V'}$, where $y_v^{V'} = 1$ if $v \in V'$ and $y_v^{V'} = 0$ otherwise, and $E' \subseteq E$ is represented by the vector $x^{E'}$, where $x_e^{E'} = 1$ if $e \in E'$ and $x_e^{E'} = 0$ otherwise. Moreover $\mathbb{0}$ and $\mathbb{1}$ are the vectors of all zeros and all ones, respectively.

Proposition 5.2. *The affine hull of P is given by*

$$\mathit{aff}(P) = \left\{(x, y) \in \mathbb{R}^{|E|+|V|} : \sum_{e \in E} x_e = n - 1\right\} \tag{5.24}$$

Proof. Let

$$p^t x + q^t y = r \tag{5.25}$$

be an equality that is satisfied by all points in P.

- Due to Lemma 5.1, for any $v \in V$ there is a spanning tree $T = (V, E_T)$ in G such that v is not branch, therefore $(x^{E_T}, \mathbb{1} \setminus y^{\{v\}})$ belongs to P. Moreover, it is easy to see that $(x^{E_T}, \mathbb{1})$ also belongs to P. This implies $q_v = 0$.

- Let $T_1 = (V, E_{T_1})$ and $T_2 = (V, E_{T_2})$ be two spanning trees in G such that $E_{T_2} = E_{T_1} \setminus \{e\} \cup \{f\}$. Such trees exist because G contains cycles. Hence $p_e = p_f$ holds, for any $e, f \in E$. Let ξ denote the value p_e, $e \in E$.

We obtain $r = \xi(n-1)$ and therefore the equality (5.25) must be a multiple of $\sum_{e \in E} x_e = n - 1$. □

Corollary 5.1. *The dimension of P is $\mathit{dim}(P) = |V| + |E| - 1$.*

The following theorem is useful to establish whether a valid inequality is a facet (see Theorem 3.6 of Nemhauser and Wolsey [42]).

Theorem 5.2. *Let $(A^=, b^=)$ be the equality set of $S \subseteq \mathbb{R}^k$ and let $F = \{x \in S : \pi x = \pi_0\}$ be a proper face of S. The following two statements are equivalent:*

- *F is a facet of S.*
- *If $\lambda x = \lambda_0$ for all $x \in F$ then*

$$(\lambda, \lambda_0) = (\alpha\pi + uA^=, \alpha\pi_0 + ub^=) \text{ for some } \alpha \in \mathbb{R} \text{ and some } u \in \mathbb{R}^{|M^=|}. \tag{5.26}$$

Proposition 5.3. *For $v \in V$, $y_v \leq 1$ defines a facet of P.*

Proof. We prove the result by showing that the conditions of Theorem 5.2 hold. Consider a fixed vertex $v \in V$. Without loss of generality, we can assume that $v = v_1$, where $V = \{v_1, \dots, v_n\}$. We can therefore write the valid inequality $y_v \leq 1$ as

$$(0, \dots, 0)x^T + (1, 0, \dots, 0)y^T \leq 1, \tag{5.27}$$

and hence $F_0^y = \{(x, y) \in P : (0, \dots, 0)x^T + (1, 0, \dots, 0)y^T = 1\}$. It is easy to see that F_0^y is a proper face. Therefore in order to prove that F_0^y represents a facet of P, from Theorem 5.2, it is sufficient to show that if $\lambda(x, y)^T = \lambda_0$ for all $(x, y) \in F_0^y$, then (λ, λ_0) can be expressed as $(\alpha\pi + uA^=, \alpha\pi_0 + ub^=)$, for some $\alpha \in \mathbb{R}$, $u \in \mathbb{R}^{|M^=|}$. As shown above, in our case $(\pi, \pi_0) = (0, \dots, 0, 1, 0, \dots, 0, 1)$, $(A^=, b^=) = (1, \dots, 1, 0, \dots, 0, n-1)$ and $|M^=| = 1$. For convenience, we represent (λ, λ_0) as

$$(\lambda, \lambda_0) = (s_1, \dots, s_m, t_1, \dots, t_n, \lambda_0).$$

Let $T_w = (V, E_{T_w})$ be a spanning tree of G such that w is not branch in T_w, where $w \in V$, $w \neq v$. It is easy to see that $(x^{E_{T_w}}, \mathbb{1} \setminus y^{\{w\}})$ and $(x^{E_{T_w}}, \mathbb{1})$ belong to F_0^y, and therefore satisfy $\lambda(x, y)^T = \lambda_0$. Consequently, $\lambda(x^{E_{T_w}}, \mathbb{1} \setminus y^{\{w\}})^T - \lambda(x^{E_{T_w}}, \mathbb{1})^T = 0$ and hence $t_w = 0$. This implies that $t_w = 0$ for any $w \in V \setminus \{v\}$.
Let $T_1 = (V, E_{T_1})$ and $T_2 = (V, E_{T_2})$ be two spanning trees of G such that $E_{T_2} = E_{T_1} \setminus (\{e\} \cup \{f\})$. Such trees exist because G contains cycles. Note that $(x^{E_{T_1}}, \mathbb{1})$ and $(x^{E_{T_2}}, \mathbb{1})$ belong to F_0^y, therefore $\lambda(x^{E_{T_1}}, \mathbb{1})^T - \lambda(x^{E_{T_2}}, \mathbb{1})^T = 0$ and hence,

through simple algebraic manipulations, we obtain $s_e = s_f$. Since T_1 is a generic spanning tree, we can conclude that $s_1 = \ldots = s_m$. From now on, we will denote this coefficient vector as s.
From these observations $\lambda(x,y)^T = \lambda_0$ becomes

$$s \sum_{e \in E} x_e + t_1 v_1 = \lambda_0. \tag{5.28}$$

Moreover for any $(x,y) \in F_0^y$, $\sum_{e \in E} x_e = n-1$ and $y_v = 1$, and hence $\lambda_0 = s(n-1) + t_1$. Therefore we obtain

$$(\lambda, \lambda_0) = (s, \ldots, s, t_1, 0, \ldots, 0, s(n-1) + t_1). \tag{5.29}$$

Note that

$$(\alpha\pi + uA^=, \alpha\pi_0 + ub^=) = (u, \ldots, u, \alpha, 0, \ldots, 0, \alpha + u(n-1)).$$

Hence, setting $\alpha = t_1$ and $u = s$ completes the proof. □

Proposition 5.4. *For $v \in V$, if all components of $G \setminus v$ are 2-disconnected, $y_v \geq 0$ defines a facet of P.*

Proof. We prove the result by showing that the conditions of Theorem 5.2 hold. Consider a fixed vertex $v \in V$. Without loss of generality, we can assume that $v = v_1$, where $V = \{v_1, \ldots, v_n\}$. We can therefore write the valid inequality $y_v \geq 0$ as

$$(0, \ldots, 0)x^T + (1, 0, \ldots, 0)y^T \geq 0, \tag{5.30}$$

and hence $F_1^y = \{(x,y) \in P : (0, \ldots, 0)x^T + (1, 0, \ldots, 0)y^T = 0\}$. It is easy to see that F_1^y is a proper face. Therefore in order to prove that F_1^y represents a facet of P, from Theorem 5.2, it is sufficient to show that if $\lambda(x,y)^T = \lambda_0$ for all $(x,y) \in F_1^y$, then (λ, λ_0) can be expressed as $(\alpha\pi + uA^=, \alpha\pi_0 + ub^=)$, for some $\alpha \in \mathbb{R}$, $u \in \mathbb{R}^{|M^=|}$. As showed above, in our case $(\pi, \pi_0) = (0, \ldots, 0, 1, 0, \ldots, 0, 0)$, $(A^=, b^=) = (1, \ldots, 1, 0, \ldots, 0, n-1)$ and $|M^=| = 1$. For convenience, we represent

(λ, λ_0) as

$$(\lambda, \lambda_0) = (s_1, \ldots, s_m, t_1, \ldots, t_n, \lambda_0).$$

Before proceeding with the proof, we need the following remark.

Remark 5.1. *As observed above, since G is 2-disconnected, the graph $G \setminus v$ can be connected or disconnected into two components. Suppose $G \setminus v$ is connected, we can distinguish two cases:*

i. $|\delta(v)| > 1$ in G. Due to Lemma 5.1, for any $w \in V \setminus \{v\}$ there exists a spanning tree $T_w = (V \setminus \{v\}, E_{T_w})$ in $G \setminus v$ such that w is not branch, therefore there exists a spanning tree $T_{v,w} = (V, E_{T_{v,w}})$ in G such that $E_{T_{v,w}} = E_{T_w} \cup \{e\}$, where $e \in \delta(v)$, $e \neq (v, w)$. $T_{v,w}$ is a spanning tree of G such that w and v are not branch.

ii. $|\delta(v)| = 1$ in G. Let $\delta(v) = \{(v, u)\}$. For any $w \in V \setminus \{v, u\}$ by the same argument as in i. there exists a spanning tree $T_{v,w}$ of G such that w and v are not branch. For any spanning tree T_u in $G \setminus v$ such that u is not branch, the degree of u in the spanning tree $T_{v,u} = (V, E_{T_{v,u}})$ in G such that $E_{T_{v,u}} = E_{T_u} \cup \{(v, u)\}$ is $|\delta_{T_{v,u}}(u)| = |\delta_{T_u}(u)| + 1$, therefore if $|\delta_{T_u}(u)| = 1$, u is not branch in $T_{v,u}$, otherwise w is branch in $T_{v,u}$. Suppose there is no spanning tree $\bar{T}_u$ in $G \setminus v$ such that $|\delta_{\bar{T}_u}(u)| = 1$, this implies that u is branch in G which yields a contradiction. Therefore there is a spanning tree $T_{v,u}$ of G such that u and v are not branch.

If $G \setminus v$ is disconnected in 2 components we can apply the same reasoning as above on each component, by distinguishing for each component $C_i(v)$, $i = 1, 2$, the cases $|\delta^i(v)| > 1$ and $|\delta^i(v)| = 1$. Therefore, for any $w \in V \setminus \{v\}$ there is a spanning tree $T_{v,w}$ of G such that w and v are not branch.

Due to Remark 5.1, for any $w \in V \setminus \{v\}$ there exists a spanning tree $T_{w,v} = (V, E_{T_{w,v}})$ in G such that v and w are not branch, therefore $(x^{E_{T_{w,v}}}, \mathbb{1} \setminus y^{\{v,w\}})$ and $(x^{E_{T_v}}, \mathbb{1} \setminus y^{\{v\}})$ belong to F_1^y, and therefore they satisfy $\lambda(x, y)^T = \lambda_0$. Consequently, $\lambda(x^{E_{T_{w,v}}}, \mathbb{1} \setminus y^{\{v,w\}})^T - \lambda(x^{E_{T_{w,v}}}, \mathbb{1} \setminus y^{\{v\}})^T = 0$. This implies $t_w = 0$ for any $w \in V \setminus \{v\}$.

Let $T_1 = (V, E_{T_1})$ and $T_2 = (V, E_{T_2})$ be two spanning trees of G such that $E_{T_2} = E_{T_1} \setminus \{e\} \cup \{f\}$ and v is not branch in the two trees. Such trees exist because G contains cycles. Note that $(x^{E_{T_1}}, \mathbb{1} \setminus y^{\{v\}})$ and $(x^{E_{T_2}}, \mathbb{1} \setminus y^{\{v\}})$ belong to F_1^y, therefore $\lambda(x^{E_{T_1}}, \mathbb{1} \setminus y^{\{v\}})^T - \lambda(x^{E_{T_2}}, \mathbb{1} \setminus y^{\{v\}})^T = 0$ and hence, through simple algebraic manipulations, we obtain $s_e = s_f$. Since T_1 is a generic spanning tree, we can conclude that $s_1 = \ldots = s_m$. From now on, we will denote this coefficient vector as s.
Due to these observations $\lambda(x, y)^T = \lambda_0$ becomes

$$s \sum_{e \in E} x_e + t_1 v_1 = \lambda_0. \tag{5.31}$$

Moreover for any $(x, y) \in F$, $\sum_{e \in E} x_e = n - 1$ and $y_v = 0$, and hence $\lambda_0 = s(n-1)$. Therefore we obtain

$$(\lambda, \lambda_0) = (s, \ldots, s, t_1, 0, \ldots, 0, s(n-1)). \tag{5.32}$$

Note that

$$(\alpha\pi + uA^=, \alpha\pi_0 + ub^=) = (u, \ldots, u, \alpha, 0, \ldots, 0, \alpha + u(n-1)).$$

Hence, setting $\alpha = 0$ and $u = s$ completes the proof.

□

Proposition 5.5. *For $e \in E$, if e is not a bridge edge in G, $x_e \leq 1$ defines a facet of P.*

Proof. We prove the result by showing that the conditions of Theorem 5.2 hold. Consider a fixed edge $e \in E$. Without loss of generality, we can assume that $e = e_1$, where $E = \{e_1, \ldots, e_m\}$. We can therefore write the valid inequality $x_e \leq 1$ as

$$(1, \ldots, 0)x^T + (0, 0, \ldots, 0)y^T \leq 1, \tag{5.33}$$

and hence $F_1^x = \{(x, y) \in P : (1, \ldots, 0)x^T + (0, 0, \ldots, 0)y^T = 1\}$. Since e is not a bridge edge in G, F_1^x is a proper face, therefore in order to prove that F_1^x represents a facet of P, from Theorem 5.2, it is sufficient to show that if $\lambda(x, y)^T = \lambda_0$ for all $(x,y) \in F_1^x$, then (λ, λ_0) can be expressed as $(\alpha\pi + uA^=, \alpha\pi_0 + ub^=)$, for some $\alpha \in \mathbb{R}$, $u \in \mathbb{R}^{|M^=|}$. As showed above, in our case $(\pi, \pi_0) = (1, 0, \dots, 0, 0, \dots, 0, 1)$, $(A^=, b^=) = (1, \dots, 1, 0, \dots, 0, n-1)$ and $|M^=| = 1$. For convenience, we represent (λ, λ_0) as

$$(\lambda, \lambda_0) = (s_1, \dots, s_m, t_1, \dots, t_n, \lambda_0).$$

Let $T = (V, E_T)$ be a spanning tree of G such that w is not branch in T_w, where $w \in V$, and e belongs to T. It is easy to see that $(x^{E_T}, \mathbb{1} \setminus y^{\{w\}})$ and $(x^{E_{T_w}}, \mathbb{1})$ belong to F_1^x, and therefore they satisfy $\lambda(x,y)^T = \lambda_0$. Consequently, $\lambda(x^{E_T}, \mathbb{1} \setminus y^{\{w\}})^T - \lambda(x^{E_T}, \mathbb{1})^T = 0$ and hence $t_w = 0$. This implies $t_w = 0$ for any $w \in V$.
Let $T_f = (V, E_{T_f})$ and $T_g = (V, E_{T_g})$ be two spanning trees of G such that $e \in E_{T_f}$ and $E_{T_g} = E_{T_f} \setminus \{f\} \cup \{g\}$, with $f \neq e$. Such trees exist because G contains cycles. Note that $(x^{E_{T_f}}, \mathbb{1})$ and $(x^{E_{T_g}}, \mathbb{1})$ belong to F_1^x, therefore $\lambda(x^{E_{T_f}}, \mathbb{1})^T - \lambda(x^{E_{T_g}}, \mathbb{1})^T = 0$ and hence, through simple algebraic manipulations, we obtain $s_f = s_g$, for all $f, g \neq e$. From now on, we will denote this coefficients as s.
Due to these observations $\lambda(x,y)^T = \lambda_0$ becomes

$$s_1 x_e + s \sum_{f \in E \setminus \{e\}} x_f = \lambda_0. \tag{5.34}$$

Moreover for any $(x,y) \in F$, $\sum_{f \in E} x_f = n-1$. Since we are assuming $x_e = 1$, then $\sum_{f \in E \setminus \{e\}} x_f = n-2$ and hence $\lambda_0 = s(n-2) + s_1$. Therefore we obtain

$$(\lambda, \lambda_0) = (s_1, s, \dots, s, 0, \dots, 0, s(n-2) + s_1). \tag{5.35}$$

Note that

$$(\alpha\pi + uA^=, \alpha\pi_0 + ub^=) = (\alpha + u, u, \dots, u, 0, \dots, 0, \alpha + u(n-1)).$$

Hence, setting $\alpha = s_1 - s$ and $u = s$ completes the proof.

□

Proposition 5.6. *For $v \in V$ and $S \subseteq \delta(v)$ with $|S| \geq 3$,*

$$\sum_{e \in S} x_e - 2 \leq (|S| - 2) y_v \tag{5.36}$$

is valid for P.

Proof. It is easy to see that for any subset S of $\delta(v)$, if more than two edges belong to the optimal solution, then vertex v has to be branch. Note that, for $S = \delta(v)$ we obtain constraints (5.4), therefore (5.36) represent a generalized version. □

Proposition 5.7. *For any cut vertex $v \in V_B$ with $\bar{c}_v = 2$, for $C_i(v)$, $i = 1, 2$ such that $|V_{C_i}(v)| \geq 2$, for $D \subseteq \delta^i(v)$ with $|D| = 2$,*

$$\sum_{e \in D} x_e \leq 1 + y_v \tag{5.37}$$

is valid for P.

Proof. Note that, v being a cut vertex with $\bar{c}_v = 2$, as stated above, at least one edge connecting v with $C_i(v)$ for $i = 1, 2$, has to be selected. As soon as a second edge connecting v with one of the two components is selected, vertex v becomes a branch vertex and y_v has to be activated. □

Proposition 5.8. *For $v \in V$ and $Q \subset \delta(v)$ such that $|Q| = |V| - 2$*

$$y_v \leq \sum_{e \in Q} x_e \tag{5.38}$$

is valid for P.

Proof. This inequality means that if there exists at least one $Q \subset \delta(v)$ such that all the edges in Q do not belong to the spanning tree, then vertex v cannot be branch. □

Proposition 5.9. *Let $R = (V_R, E_R)$ be a cycle of cardinality three, i.e. $V_R = \{a, b, c\}$ and $E_R = \{f_{ab}, f_{ac}, f_{bc}\}$. For $v \in V_R$ such that $|\delta(v)| = 3$, without loss of generality assume that $v = a$,*

$$y_a + x_{f_{bc}} \leq 1 \tag{5.39}$$

is valid for P. Moreover, if there exist at least two vertices a and b in the cycle having degree 3 in the graph, then

$$y_a + y_b \leq x_{f_{ab}} \tag{5.40}$$

is valid for P. Finally, if the three vertices all have degree 3, then

$$y_a + y_b + y_c \leq 1. \tag{5.41}$$

Proof. Constraints (5.39) state that if a is branch, then the edge f_{bc} cannot be selected for otherwise the solution would contain a cycle. Conversely, if edge f_{bc} belongs to the solution, then a will not be a branch vertex. Constraints (5.40) impose that only one vertex between a and b can be branch whenever edge f_{ab} is selected. If the three vertices have degree 3, then constraints (5.41) state that at most one of them can be a branch vertex. $\square$

5.4 Directed and hybrid reformulations

In the combinatorial optimization literature, problems originally defined over undirected graphs are often reformulated over corresponding directed graphs. In this section we consider a directed integer programming reformulation (DILP) of MBVP as a spanning arborescence problem. To develop a model for this directed version of the problem, we fix an arbitrary vertex $r \in V$ as the root vertex and we consider the directed graph $D = (V, A)$ obtained by replacing each edge $(v, u) \in E$ by arcs (v, u) and (u, v) in A. In addition to the previously defined variables y_v, $v \in V$, for each arc $a \in A$, we define z_a as a binary variable equal to 1 if and only if arc a belongs to the spanning arborescence. In association with graph D, we define $\delta^+(w) = \{(v, u) \in A : v = w\}$ and $\delta^-(w) = \{(v, u) \in A : u = w\}$. Note that, from the definition of A, it follows that $|\delta^+(v)| = |\delta^-(v)| = |\delta(v)|$, $v \in V$. The DILP formulation is then

$$\text{minimize } z = \sum_{v \in V} y_v \tag{5.42}$$

subject to

$$\sum_{e\in A(S)} z_a \leq |S|-1 \qquad S\subset V,\ |S|>2 \tag{5.43}$$

$$\sum_{a\in A} z_a = n-1 \tag{5.44}$$

$$\sum_{a\in\delta^-(v)} z_a = 1 \qquad v\in V\setminus\{r\} \tag{5.45}$$

$$\sum_{a\in\delta^+(v)} z_a - 1 \leq (|\delta(v)|-2)y_v \qquad v\in V\setminus\{r\} \tag{5.46}$$

$$\sum_{a\in\delta^+(r)} z_a - 2 \leq (|\delta(r)|-2)y_r \tag{5.47}$$

$$2y_v \leq \sum_{a\in\delta^+(v)} z_a \qquad v\in V\setminus\{r\} \tag{5.48}$$

$$2y_r \leq \sum_{a\in\delta^+(r)} z_a - 1 \tag{5.49}$$

$$z_a \in \{0,1\} \qquad a\in A \tag{5.50}$$

$$y_v \in \{0,1\} \qquad v\in V. \tag{5.51}$$

Constraints (5.43), (5.44) and (5.50) characterize the spanning arborescence polytope. Note that the inequalities (5.4) and (5.7), for the undirected graph formulation, are split into inequalities (5.46), (5.47) and (5.48), (5.49), respectively, for the directed graph reformulation. Also observe that due to (5.45), one unit is subtracted in the left-hand side of (5.46) instead of two units in the corresponding inequalities (5.4).

It is easy to see that several of the properties described for the undirected formulation are easily adaptable to the directed case. Moreover, with the only exception of the root vertex r, no more than one outwards pointing arc may be incident to a no branch vertex. Hence the inequalities

$$\sum_{a\in W} z_a - 1 \leq (|W|-1)y_v \qquad v\in V\setminus\{r\},\ W\subset\delta^+(v) : |W|\geq 2 \tag{5.52}$$

are clearly valid for the directed formulation. Now, let P_{STP} denote the spanning tree polytope, that is the linear relaxation of (5.5) plus constraints (5.2) and (5.3) and let D_{STP} denote the polytope defined by the linear relaxation of (5.50 plus constraints (5.43)−(5.45) and

$$x_e = z_{vu} + z_{uv} \qquad e = (v, u) \in E, \tag{5.53}$$

in the x-space. Moreover, let D_L denote the polytope defined by the intersection of D_{STP} with the linear relaxation of (5.51) plus (5.46)−(5.49), in the x-space.

Proposition 5.10. *The undirected and the directed formulation for the Minimum Branch Vertex Spanning Tree Problem are equivalent if constraints (5.53) are introduced in the DILP model.*

Proof. D_{STP} yields an alternative description of the spanning tree polytope P_{STP}, hence $P_{STP} = D_{STP}$ (see [36] for the details). Note that summing up (5.45) and (5.46) we obtain (5.4), and summing up (5.45) and (5.48) we obtain (5.7). Therefore, $D_L \subseteq P_L$. Conversely, suppose $(\bar{x}, \bar{y})$ is feasible in P_L, then $\bar{x}$ satisfies (5.2) and (5.3), therefore there exists $\bar{z}$ that satisfies (5.43) − (5.45), for an arbitrary $r \in V$. For any $v \neq r$ we can rewrite (5.4) and (5.7) as follows:

$$\sum_{a\in\delta^+(v)} z_a + \sum_{a\in\delta^-(v)} z_a - 2 \leq (|\delta(v)| - 2)y_v \qquad v \in V \setminus \{r\} \tag{5.54}$$

$$2y_v \leq \sum_{a\in\delta^+(v)} z_a + \sum_{a\in\delta^-(v)} z_a - 1 \qquad v \in V \setminus \{r\} \tag{5.55}$$

Note that (5.54) and (5.45) imply (5.46), and that (5.55) and (5.45) imply (5.48). Then $(\bar{x}, \bar{y}, \bar{z})$ is feasible in D_L, and hence $P_L = D_L$. □

The set defined by constraints (5.2)−(5.7), (5.45)−(5.47), (5.50) and (5.53) define the set of feasible solutions for the MBVP. We refer to it as the *hybrid reformulation* (HILP). Note that we keep constraints (5.4) in this formulation even if they are redundant in the presence of constraints (5.45)−(5.47), (5.50) and (5.53). However the formulation is still correct and test results show that this choice helps CPLEX to find an optimal solutions faster. Let H_L denote the polytope obtained by the linear relaxation of (5.5), (5.6) and (5.50) plus (5.2)−(5.4),(5.7), (5.45)−(5.47) and (5.53).

5.5 Branch-and-cut algorithm

We solve the MBVP by means of a branch-and-cut algorithm which is summarized in Algorithm 4. Before executing the algorithm we reduce the graph in a preprocessing phase by exploiting the properties introduced in Section 5.2.1. In line 1, an initial feasible solution is identified by searching a minimum spanning tree using Prim's algorithm [45]. With any edge $e = (v, u)$ we associate weight $w_e = n$ if $\min\{|\delta(v)|, |\delta(u)|\} \leq 2$, otherwise $w_e = n - \max\{|\delta(v)|, |\delta(u)|\}$. In line 3, the first

Algorithm 4: Branch-and-cut algorithm

```
Input: integer program $P$.
Output: an optimal solution of $P$.
1  Identify initial feasible solution $T_0$. Get number $b_0$ of branch vertices in $T_0$
2  $ub \leftarrow b_0$, $L = \emptyset$
3  Define a first subproblem and insert it in the list $L$
4  while $L$ is not empty do
5      chose the subproblem and remove it from $L$
6      solve the subproblem to obtain the lower bound $lb$
7      if $lb < ub$ then
8          if the solution is integer then
9              if the solution is feasible then
10                 $ub \leftarrow lb$
11                 update incumbent solution
12             else
13                 search and add SEC on integer solutions
14         else
15             search violated constraints
16             if root node then
17                 search SEC on non-integer solutions
18             if violated constraints are identified then
19                 add them to the model
20             else
21                 branch on a variable and add the corresponding subproblems
                   in $L$
```

subproblem is obtained by relaxing the subtour elimination constraints (5.2), except

for the case where $|S| = 3$, as well as the integrality constraints on the variables. We also identify all the bridges, the cocycles and the cut vertices of the graph and we add the correspondent constraints (5.16), (5.19) and (5.20). In line 13, a search for violated constraints (5.2) is performed on the integer solutions by identifying the connected components and by adding the subtour elimination constraints induced by the subsets of vertices of all the components containing at least one cycle. In line 17, at a non-integer solution, constraints (5.2) are separated using the max-flow algorithm proposed by Padberg and Wolsey [44]. The max-flow obtained with this algorithm is $f = |\bar{S}| - \sum_{e \in E(\bar{S})} x_e + C$, where $\{\bar{S}, V \setminus \bar{S}\}$ represents the cut-set associated to the max-flow and C is a constant value depending on the vertex set V, therefore a constraint is violated if $f - C$ is less than 1. Since we are not interested in constraints with a small violation, a constraint is generated whenever $f - C$ is less than $1 - \epsilon$, for a fixed ϵ depending on the instances. For the non-integer solutions, we run the max-flow procedure only on the root node.
The branch-and-cut algorithm was applied to both undirected and hybrid formulations. In the first case, in line 15, a search for violated inequalities (5.36) and (5.37) is performed. Valid inequalities (5.38), (5.39), (5.40) and (5.41) turned out to be ineffective and were not considered. A subset of the most violated inequalities (5.37) is added to the cut-pool. The separation procedure used for inequalities (5.36) is that of Lucena et al. ([35]). Let $(\bar{x}, \bar{y})$ be a feasible solution for the linear programming relaxation, and for every $v \in V$ such that $|\delta(v)| \geq 3$, order the elements in $\{\bar{x}_e : e \in \delta(v)\}$ in decreasing value. Then, for $(\bar{x}, \bar{y})$, $v \in V$, and every $k \in \{3, \ldots, |\delta(v)| - 1\}$, compute $\sum_1^k \bar{x}_{e_k} - (k-2)\bar{y}_v$. This procedure identifies a set S of cardinality k with the largest value for the left-hand side of (5.36) for vertex v. If this value is greater than 2, it has identified the most violated inequality, otherwise, no violated inequality (5.36) exists for v. For any vertex $v \in V$, having $|\delta(v)| \geq 3$, we first consider all $S \subseteq \delta(v)$ such that $|S| = 3$ and we add a subset of the most violated inequalities (5.36) by the current relaxed solution. Moreover, we run the procedure previously described for $k \in \{4, \ldots, |\delta(v)| - 1\}$ and we add at most one violated constraint for each value of k. In line 19 all the violated constraints identified are added to the model. In the implementation for the hybrid formulation, in line 15, a search for violated inequalities (5.52) is also performed. The separation procedure is the same described for inequalities (5.36). As for the

previous case, we first look for all subsets $W \subset \delta^+(v)$ such that $|W| = 2$ and a subset of the most violated inequalities is added to the cut-pool, then the separation procedure is performed for $k \geq 3$. In line 21, branching takes place in priority on the y_v variables.

5.6 Computational results

The branch-and-cut algorithm was coded in C and solved using IBM ILOG CPLEX 12.5.1. The computational experiments were performed on a 64-bit GNU/Linux operating system, 96 GB of RAM and one processor Intel Xeon X5675 running at 3.07 GHz.
In our tests the *MIPEmphasis* parameter is set on the best bound value and the others parameters as default. For all the instances the constant ϵ introduced to identify violated constrains (5.2) on the non-integer solutions is set equal to 0.7.
Experiments for the MBVP were conducted on benchmark instances. Carrabs et al. [6] generated instances with n between 20 and 1000 and different densities. Note that dense graphs often can contain a Hamiltonian path, therefore the authors generated sparse graphs. These instances were also used by Marín [37]. In his paper the author divides the instances into two groups: medium instances (with $n \leq 500$) and large instances (with $n \geq 600$). Here we call *small* the instances with $n \leq 200$, *medium* those with $250 \leq n \leq 500$ and *large* those with $n \geq 600$.

Table 5.1 and 5.2 report the results for the undirected formulation applied to the small and medium instances. In the tables each line represents an average over five instances having the same number of vertices and of edges. In both tables the first two columns represent the instances, columns **ub**, **opt** and **sec** report the average of the upper bounds found with Prim's algorithm [45], the average of the optimal solution values and the average of the computational time needed to compute them. Moreover, whenever α instances of a group are not solved to optimality within the time limit of one hour, we write (α) appears close to the solution value. The numbers of bridges, cocycles and cut vertices are also reported. Columns **nodes** and **cuts** represent the number of nodes in the search tree and the number of cuts added.
Results for small, medium and large instances for the hybrid formulation are

n	m	ub	opt	bridge	cocycle	cut vertex	nodes	cuts	sec
20	27	4.6	2.4	6.2	5.6	4.2	0.0	3.2	0.0
20	34	5.4	1.2	2.4	5.0	2.2	0.0	5.2	0.0
20	42	3.6	0.2	0.8	2.2	0.8	0.0	0.0	0.0
20	49	3.6	0.0	0.0	1.2	0.0	0.0	0.4	0.0
20	57	3.0	0.0	0.2	0.0	0.2	0.0	1.8	0.0
40	50	12.2	7.4	16.2	13.4	9.2	0.0	15.8	0.0
40	60	9.2	3.4	7.4	13.6	5.6	0.4	22.2	0.0
40	71	10.8	1.6	5.2	7.4	4.6	0.0	18.2	0.0
40	81	8.4	0.8	2.2	6.6	2.2	0.0	19.4	0.0
40	92	8.2	0.6	2.2	4.4	2.2	0.0	13.8	0.0
60	71	19.6	13.0	28.4	27.6	15.6	0.0	16.0	0.0
60	83	18.0	8.2	17.8	21.0	11.6	1.8	56.0	0.1
60	95	15.4	5.4	12.0	18.8	9.8	25.0	189.8	0.4
60	107	15.6	3.4	7.2	13.2	6.4	1.0	126.4	0.2
60	119	12.8	1.6	4.8	11.6	4.8	7.6	151.6	0.3
80	93	24.0	16.4	40.8	35.0	21.2	1.6	27.0	0.1
80	106	23.6	12.0	27.4	30.0	17.4	7.0	77.8	0.1
80	120	22.4	8.8	19.4	23.4	13.8	36.4	195.4	0.5
80	133	21.0	5.6	12.4	25.0	10.6	12.8	186.6	0.8
80	147	18.6	3.4	9.8	19.2	8.4	17.2	199.6	0.4
100	114	31.6	23.8	56.8	38.6	27.4	4.2	25.4	0.1
100	129	32.0	16.4	38.6	35.6	22.4	8.0	109.2	0.5
100	144	29.8	11.8	26.2	32.2	18.0	18.8	189.2	0.6
100	159	27.4	8.4	18.6	32.4	14.8	47.4	334.2	1.1
100	174	24.4	6.2	15.4	25.2	11.8	4937.0	2220.6	126.9
120	136	39.6	29.6	69.8	45.6	33.4	10.4	36.4	0.1
120	152	38.8	21.8	48.4	48.4	27.8	19.2	124.4	0.4
120	169	34.6	16.0	36.4	38.4	23.2	28.6	214.4	0.8
120	185	33.2	11.6	25.4	41.4	18.2	162.0	455.4	1.8
120	202	31.8	8.6	20.4	34.6	15.0	93.4	442.8	2.3
140	157	45.4	34.2	79.8	71.0	38.6	14.0	64.0	0.3
140	175	43.6	25.8	59.0	57.6	33.4	15.4	141.6	0.7
140	193	40.6	18.8	41.6	52.8	28.4	124.8	329.8	1.8
140	211	39.2	15.2	35.6	41.2	24.0	128.2	466.4	1.9
140	229	36.0	10.6	23.8	43.0	19.2	253.0	750.4	4.8
160	179	52.6	39.8	94.0	64.6	44.8	0.0	28.8	0.2
160	198	49.4	31.2	69.2	68.2	37.8	45.6	179.8	1.1
160	218	47.2	23.4	50.2	62.8	31.2	112.6	359.2	1.9
160	237	44.6	17.4	39.4	50.8	27.4	198.0	542.6	2.9
160	257	44.0	13.4	32.2	43.0	24.8	248.4	799.0	6.6
180	200	58.6	46.4	111.6	76.4	51.4	12.0	52.4	0.4
180	221	55.6	35.0	79.8	67.0	44.2	99.6	215.0	1.5
180	242	54.2	25.4	58.8	69.6	37.0	204.4	491.2	3.7
180	263	53.2	21.0	46.6	61.0	32.4	805.0	905.8	14.5
180	284	47.6	17.6	39.4	56.0	29.6	528.2	1100.8	11.9
200	222	63.6	50.6	127.8	74.8	57.0	14.6	69.4	0.6
200	244	62.0	39.4	92.4	77.8	49.6	49.8	174.0	1.3
200	267	59.4	30.4	69.0	72.0	40.2	130.8	390.0	3.7
200	289	56.4	24.8	56.8	68.8	38.4	2166.4	1185.8	56.6
200	312	57.2	(1)25.8	42.2	57.6	30.2	5464.6	3942.2	732.5

Table 5.1: Undirected formulation: computational results for small instances

n	m	ub	opt	bridge	cocycle	cut vertex	nodes	cuts	sec
250	273	81.4	66.0	164.4	100.2	71.4	1.4	51.0	0.8
250	297	78.6	53.0	120.8	110.8	60.8	312.0	318.8	5.0
250	321	75.8	43.4	101.8	93.8	57.6	277.4	514.8	7.4
250	345	74.6	34.4	76.2	90.0	47.8	1616.2	930.2	47.9
250	369	70.8	26.2	60.0	85.2	40.2	732.4	1352.8	42.7
300	326	97.4	81.0	203.0	121.8	87.4	30.2	127.2	1.9
300	353	95.0	67.8	160.2	116.4	78.6	171.4	323.6	6.2
300	380	92.6	54.6	124.8	114.0	69.0	572.4	785.4	21.9
300	407	89.6	46.2	104.6	103.4	61.8	1808.0	1619.6	75.9
300	434	85.0	37.2	86.4	89.4	56.2	1657.2	1933.0	143.8
350	378	113.4	94.6	238.8	143.2	102.8	70.2	152.6	5.0
350	406	111.6	80.6	190.0	145.6	93.6	452.4	476.8	10.1
350	435	108.0	65.6	151.0	150.8	84.4	2016.2	1379.4	85.6
350	463	107.2	56.6	124.2	128.4	75.8	11731.0	1945.2	663.4
350	492	102.6	45.4	103.6	123.8	67.2	5569.6	2322.0	444.0
400	429	130.8	111.8	282.6	167.2	119.6	56.2	123.6	3.9
400	459	128.0	94.0	226.4	165.0	109.4	851.6	782.6	21.0
400	489	126.2	(1)88.4	184.8	152.4	99.0	2315.8	5068.8	742.9
400	519	122.2	68.4	154.2	154.4	88.4	9878.8	2517.8	979.4
400	549	118.4	56.0	131.2	141.2	80.2	3204.6	2962.6	350.2
450	482	148.6	125.8	318.6	177.8	135.4	33.6	116.0	4.8
450	515	146.0	107.4	250.6	202.8	121.6	1298.6	846.2	75.4
450	548	140.0	90.4	208.8	184.2	110.4	3686.0	4059.0	835.4
450	581	139.2	(1)77.6	176.6	167.8	100.4	12719.2	2901.4	1363.9
450	614	133.2	(3)66.4	151.8	153.8	93.8	17717.4	3686.4	2766.6
500	534	164.6	141.6	361.0	191.2	150.6	70.4	149.2	10.2
500	568	160.8	120.8	294.2	187.0	137.2	948.6	770.0	53.8
500	603	158.2	105.6	246.0	198.4	126.8	3089.4	1981.2	260.3
500	637	151.6	(2)117.2	210.6	181.2	116.8	4850.8	6615.4	1902.9
500	672	148.4	(5)122.8	170.0	194.6	104.4	13407.2	11163.8	3600.0

Table 5.2: Undirected formulation: computational results for medium instances

n	m	ub	opt	bridge	cocycle	cut vertex	nodes	cuts	sec
20	41.8	4.0	0.8	1.9	2.8	1.5	0.0	1.8	0.0
40	70.8	9.8	2.8	6.6	9.1	4.8	0.2	33.6	0.1
60	95.0	16.3	6.3	14.0	18.4	9.6	0.0	67.4	0.5
80	119.8	21.9	9.2	22.0	26.5	14.3	1.0	83.9	0.7
100	144.0	29.0	13.3	31.1	32.8	18.9	1.7	108.7	1.0
120	168.8	35.6	17.5	40.1	41.7	23.5	2.6	135.0	1.1
140	193.0	41.0	20.9	48.0	53.1	28.7	6.2	178.8	2.0
160	217.8	47.6	25.0	57.0	57.9	33.2	2.8	165.6	1.9
180	242.0	53.8	29.1	67.2	66.0	38.9	9.0	212.3	2.5
200	266.8	59.7	32.6	77.6	70.2	43.1	6.8	213.4	3.1
250	321.0	76.2	44.6	104.6	96.0	55.6	5.8	209.8	3.1
300	380.0	91.9	57.4	135.8	109.0	70.6	6.0	230.2	4.2
350	434.8	108.6	68.6	161.5	138.4	84.8	7.9	298.8	6.9
400	489.0	125.1	81.8	195.8	156.0	99.3	21.0	355.2	9.1
450	548.0	141.4	93.4	221.3	177.3	112.3	17.5	333.7	9.5
500	602.8	156.7	106.7	256.4	190.5	127.2	10.3	332.0	9.8

Table 5.3: Hybrid formulation: computational results for small and medium instances

n	m	ub	opt	bridge	cocycle	cut vertex	nodes	cuts	sec
600	637	197.6	183.8	493.6	68.8	188.0	0.0	74.0	3.2
600	674	192.6	167.2	437.4	71.6	176.4	0.0	148.8	8.7
600	712	188.0	150.6	394.4	68.6	168.6	1.6	229.0	10.3
600	749	182.2	138.8	363.4	55.6	161.0	21.2	335.6	17.6
600	787	173.8	125.8	333.6	49.4	153.2	18.2	333.8	16.2
700	740	232.0	214.4	576.8	91.4	218.6	0.0	100.4	8.7
700	780	224.8	198.0	518.4	89.2	206.8	2.6	176.6	11.0
700	821	218.0	180.0	470.2	79.4	198.2	0.6	257.2	12.5
700	861	212.4	164.0	436.6	62.8	191.4	3.2	291.0	17.4
700	902	205.0	154.2	403.2	63.6	183.2	1.0	293.6	14.7
800	843	265.4	245.6	666.8	90.6	252.2	0.0	102.0	10.3
800	886	256.8	227.6	599.4	98.8	237.4	1.8	169.0	11.2
800	930	253.6	208.4	546.6	89.2	228.8	10.2	321.6	22.7
800	973	245.2	194.2	505.8	82.0	221.4	72.4	658.4	48.8
800	1017	232.2	176.2	468.2	71.4	212.8	23.8	479.8	37.1
900	944	300.6	279.6	756.4	105.4	284.8	0.0	118.4	12.6
900	989	290.0	259.2	685.6	110.4	271.4	188.8	339.6	66.2
900	1034	286.6	240.6	633.0	105.0	262.2	28.2	405.4	30.2
900	1079	281.4	223.2	583.6	98.0	251.2	12.6	489.8	90.5
900	1124	269.0	206.0	547.6	83.2	242.4	2.0	372.0	30.7
1000	1047	332.6	312.0	849.6	110.2	317.0	8.4	148.8	26.2
1000	1095	323.2	290.0	767.0	121.0	303.2	0.0	209.2	17.0
1000	1143	318.6	271.2	705.0	121.2	290.2	74.2	613.4	57.1
1000	1191	310.4	251.0	657.6	109.8	279.8	53.6	621.2	75.4
1000	1239	303.8	235.2	609.8	105.6	268.4	45.8	735.4	62.6

Table 5.4: Hybrid formulation: computational results for large instances

reported in Table 5.3 and Table 5.4. In Table 5.3 each line represent an average over 25 instances having the same number of vertices. The table reports the results for both small and medium instances. In Table 5.4 each line represents an average over five instances having the same number of vertices and edges. The two tables have the same structure described above. Note that in this case, all the instances were solved optimally. Our experimental results show that the hybrid formulation is more efficient and faster. It allows us to solve all the small and medium instances within less than 10 seconds, while the undirected formulation could not find an optimal solution on 13 instances after one hour. Moreover, we can solve all the large instances, up to $n = 1000$ within an average time of 90.5 seconds. Finally, note that the number of nodes in the search tree is relatively small and all families of cuts are useful.

5.6.1 LP lower bounds and duality gaps

The purpose of this subsection is to present the LP lower bounds obtained by adding one valid inequality each time to the hybrid formulation.

In Table 5.5 each line is an average over 25 instances, while in Table 5.6 the

n	m	**opt**	$w(H_L)$	$w(H_L^1)$	$w(H_L^2)$	$w(H_L^3)$	$w(H_L^4)$	$w(H_L^5)$
20	41.8	0.8	0.59	0.65	0.64	0.60	0.61	0.62
40	70.8	2.8	2.16	2.31	2.29	2.21	2.23	2.25
60	95.0	6.3	5.18	5.64	5.56	5.33	5.42	5.44
80	119.8	9.2	7.79	8.44	8.25	8.05	8.19	8.10
100	144.0	13.3	11.75	12.47	12.28	12.06	12.23	12.16
120	168.8	17.5	15.60	16.46	16.24	16.02	16.20	16.14
140	193.0	20.9	18.64	19.64	19.36	19.18	19.47	19.17
160	217.8	25.0	22.74	23.75	23.55	23.25	23.55	23.31
180	242.0	29.1	26.39	27.62	27.30	27.08	27.41	27.06
200	266.8	32.6	30.00	31.22	30.91	30.55	30.94	30.68
250	321.0	44.6	41.72	43.06	42.68	42.43	42.96	42.42
300	380.0	57.4	53.74	55.55	54.93	54.73	55.23	54.51
350	434.8	68.6	63.96	65.93	65.36	65.40	66.16	65.00
400	489.0	81.8	77.31	79.33	78.68	78.96	79.62	78.21
450	548.0	93.4	88.32	90.66	89.75	90.04	90.79	89.29
500	602.8	106.7	101.75	104.26	103.43	103.48	104.16	102.86

Table 5.5: Hybrid formulation: lower bounds for MBVP on small and medium instances

average is computed over five instances. In both the tables the first two columns represent the instances and the third one the optimal solution. The next columns provide lower bounds $w(H_L)$, $w(H_L^1)$, $w(H_L^2)$, $w(H_L^3)$, $w(H_L^4)$ and $w(H_L^5)$, where H_L denotes the polytope obtained by relaxing the integrality constraints in the hybrid formulation, while H_L^1 and H_L^2 denote the intersection of H_L with (5.52) for $W \subset \delta^+(v)$ such that $|W| = 2$ and $|W| \geq 3$, respectively. Moreover, H_L^3 denotes the intersection of H_L with (5.37), while H_L^4 and H_L^5 with (5.36) for $S \subseteq \delta(v)$ such that $|S| = 3$ and $|S| \geq 4$, respectively. It is easy to see from the tables that all the cuts help improve the lower bound, in particular $w(H1)$ and $w(H4)$ seem to yield the best lower bounds in most cases. Inequalities (5.52) for $W \subset \delta^+(v)$ such that $|W| = 2$ and (5.36) for $S \subseteq \delta(v)$ such that $|S| = 3$ are the most useful cuts. This is evident in Table 5.7 and 5.8 which provide the duality gap with respect to the optimal solution on the six polytopes.

n	m	**opt**	$w(H_L)$	$w(H_L^1)$	$w(H_L^2)$	$w(H_L^3)$	$w(H_L^4)$	$w(H_L^5)$
600	637	183.8	180.40	180.94	180.70	182.03	182.79	180.63
600	674	167.2	163.83	164.63	164.47	164.97	166.11	164.45
600	712	150.6	147.24	148.31	147.99	148.10	148.96	147.73
600	749	138.8	136.09	136.97	136.82	136.50	136.83	136.75
600	787	125.8	123.87	124.66	124.66	124.17	124.46	124.85
700	740	214.4	211.03	211.56	211.30	212.89	213.68	211.08
700	780	198.0	193.67	194.79	194.56	195.11	196.52	194.30
700	821	180.0	175.72	177.14	176.91	177.11	178.28	176.56
700	861	164.0	160.81	161.82	161.73	161.29	161.87	161.77
700	902	154.2	151.16	152.43	152.42	151.71	152.17	152.50
800	843	245.6	242.02	242.55	242.25	244.04	245.08	242.17
800	886	227.6	223.44	224.24	224.15	224.82	226.47	223.67
800	930	208.4	204.26	205.36	205.26	205.55	206.92	204.82
800	973	194.2	189.87	191.48	191.15	190.73	191.68	191.12
800	1017	176.2	172.37	173.72	173.63	172.99	173.49	173.79
900	944	279.6	275.15	275.76	275.33	277.72	278.88	275.17
900	989	259.2	253.97	255.29	254.81	256.15	257.71	254.26
900	1034	240.6	235.66	236.88	236.84	237.38	239.09	236.37
900	1079	223.2	218.02	219.98	219.59	219.52	220.61	219.09
900	1124	206.0	202.19	203.78	203.50	202.87	203.41	203.47
1000	1047	312.0	307.48	308.28	307.97	310.08	311.33	307.63
1000	1095	290.0	283.03	284.62	284.23	286.72	288.50	283.78
1000	1143	271.2	265.37	266.94	266.76	267.40	269.43	266.37
1000	1191	251.0	244.99	246.92	246.68	246.82	248.22	246.29
1000	1239	235.2	230.27	232.16	231.90	231.10	231.92	231.74

Table 5.6: Hybrid formulation: lower bounds for MBVP on large instances

n	m	**opt**	$G_{H_L}(\%)$	$G_{H_L^1}(\%)$	$G_{H_L^2}(\%)$	$G_{H_L^3}(\%)$	$G_{H_L^4}(\%)$	$G_{H_L^5}(\%)$
20	41.8	0.8	29.1	16.8	17.9	26.4	25.6	23.3
40	70.8	2.8	28.0	19.5	20.7	24.8	23.9	22.4
60	95.0	6.3	22.1	12.1	13.6	18.5	16.6	16.3
80	119.8	9.2	18.6	9.5	11.9	14.7	12.9	14.1
100	144.0	13.3	13.4	6.8	8.5	10.4	8.9	9.5
120	168.8	17.5	12.3	6.4	7.9	9.4	8.2	8.6
140	193.0	20.9	12.3	6.5	8.1	9.1	7.5	9.1
160	217.8	25.0	10.1	5.4	6.3	7.7	6.3	7.4
180	242.0	29.1	10.2	5.3	6.5	7.4	6.1	7.5
200	266.8	32.6	8.8	4.5	5.6	6.8	5.5	6.4
250	321.0	44.6	6.9	3.6	4.5	5.1	3.8	5.1
300	380.0	57.4	6.7	3.3	4.4	4.8	3.9	5.2
350	434.8	68.6	7.2	4.0	4.9	4.8	3.6	5.5
400	489.0	81.8	5.9	3.2	4.0	3.6	2.8	4.6
450	548.0	93.4	5.7	3.0	4.0	3.7	2.8	4.6
500	602.8	106.7	4.9	2.4	3.2	3.1	2.5	3.8

Table 5.7: Hybrid formulation: duality gap on small and medium instances

n	m	**opt**	$G_{H_L}(\%)$	$G_{H_L^1}(\%)$	$G_{H_L^2}(\%)$	$G_{H_L^3}(\%)$	$G_{H_L^4}(\%)$	$G_{H_L^5}(\%)$
600	637	183.8	1.9	1.6	1.7	1.0	0.6	1.8
600	674	167.2	2.1	1.6	1.7	1.4	0.7	1.7
600	712	150.6	2.3	1.5	1.8	1.7	1.1	1.9
600	749	138.8	2.0	1.3	1.4	1.7	1.4	1.5
600	787	125.8	1.6	0.9	0.9	1.3	1.1	0.8
700	740	214.4	1.6	1.3	1.5	0.7	0.3	1.6
700	780	198.0	2.2	1.6	1.8	1.5	0.8	1.9
700	821	180.0	2.4	1.6	1.7	1.6	1.0	1.9
700	861	164.0	2.0	1.3	1.4	1.7	1.3	1.4
700	902	154.2	2.0	1.2	1.2	1.6	1.3	1.1
800	843	245.6	1.5	1.3	1.4	0.6	0.2	1.4
800	886	227.6	1.9	1.5	1.5	1.2	0.5	1.8
800	930	208.4	2.0	1.5	1.5	1.4	0.7	1.7
800	973	194.2	2.3	1.4	1.6	1.8	1.3	1.6
800	1017	176.2	2.2	1.4	1.5	1.9	1.6	1.4
900	944	279.6	1.6	1.4	1.6	0.7	0.3	1.6
900	989	259.2	2.1	1.5	1.7	1.2	0.6	1.9
900	1034	240.6	2.1	1.6	1.6	1.4	0.6	1.8
900	1079	223.2	2.4	1.5	1.6	1.7	1.2	1.9
900	1124	206.0	1.9	1.1	1.2	1.5	1.3	1.2
1000	1047	312.0	1.5	1.2	1.3	0.6	0.2	1.4
1000	1095	290.0	2.5	1.9	2.0	1.1	0.5	2.2
1000	1143	271.2	2.2	1.6	1.7	1.4	0.7	1.8
1000	1191	251.0	2.5	1.7	1.8	1.7	1.1	1.9
1000	1239	235.2	2.1	1.3	1.4	1.8	1.4	1.5

Table 5.8: Hybrid formulation: duality gap on large instances

Conclusions and Future Work

In this thesis we have modeled and solved three combinatorial optimization problems defined on graphs. The *Rainbow Cycle Cover Problem* and the *Rainbow Spanning Forest Problem* are defined on edge-colored graph and the objective is partitioning the edges of the input graph in the minimum number of rainbow cycles and trees, respectively. The *Minimum Branch Vertices Spanning Tree Problem* belongs to the class of $\mathcal{NP}$-hard spanning tree problems. The main contributions can be summarized as follows.

The Rainbow Cycle Cover Problem (Chapter 3)

We have proposed a mathematical formulation of the Rainbow Cycle Cover Problem, some properties that a rainbow cycle cover must satisfied and some valid inequalities used to solve the RCCP within a branch-and-cut algorithm. Computational experiments were conducted on randomly generated instances. Results show that the branch-and-cut algorithm is able to solve instances having between 20 and 50 vertices, and between 3 and 18 colors. The presence of a constant M in the objective function and the symmetry of the problem affect the final results and the effectiveness of the algorithm, mainly on instances with a large number of edges and a small number of colors when the number of vertices increases.

The Rainbow Spanning Forest Problem (Chapter 4)

We have proved that the Rainbow Spanning Forest Problem is $\mathcal{NP}$-complete on acyclic graphs and we have provided a polynomial case. Furthermore, we have proposed two mathematical formulations of the problem, (ILP1) and (ILP2), and some valid inequalities for the second one. To solve large instances a greedy algorithm and a multi-start scheme were introduced. Computational experiments were conducted on randomly generated instances on the first formulation, the greedy algorithm and the multi-start scheme. The 80% of the small scenarios were solved by ILP1, within the time limit, and this percentage is close to 92% for the small scenarios with a density at least equal to 0.2. Moreover, the running time of ILP1 decreases as the density increases. Indeed, all the scenarios with $d = 0.3$ are solved within less than 340 seconds by ILP1. Thanks to the optimal solutions provided by the mathematical model on the small scenarios, we certified the effectiveness of GA and MS algorithms. This last algorithm is the most effective with eight optimal solutions found and a gap from the best known solution only two times greater than one. The negligible running time of both algorithms make them particularly suitable to be embedded in exact approaches where it is often required to quickly generate good solutions. On the large scenarios the MS algorithm results slower than the GA algorithm but its solution are much better.

The Minimum Branch Vertices Spanning Tree Problem (Chapter 5)

We have modeled and solved the Minimum Branch Vertices Spanning Tree Problem. We have provided two mathematical formulations based on an undirected and on a directed graph, respectively, and a hybrid formulation obtained by merging the first two models. Moreover, we have derived some properties and some valid inequalities for the problem. A branch-and-cut approach was proposed on the undirected and on the hybrid formulations. Results show that the hybrid formulation is superior to the undirected formulation and that our branch-and-cut algorithm applied to it solves all benchmark instances to optimality.

Future research should be conducted to extend our results. Given the hardness of the Rainbow Cycle Cover Problem and its symmetry, the exact approach has proved to be naturally limited in the size of the instances on which it can be applied, therefore a further effort in this direction could be to study heuristic approaches to solve the problem. A possible direction for future work on the Rainbow Spanning Forest Problem could be the development of metaheuristics that are faster and more efficient than MS. Moreover, a branch-and-cut algorithm on the second mathematical formulation, where to use the valid inequalities proposed, could be implemented and a polyhedral analysis to derive polyhedral results could be conducted.

References

[1] R. Bhatia, S. Khuller, R. Pless, and Y. Sussmann. The full degree spanning tree problem. *Networks*, 36:203–209, 200.

[2] J. A. Bondy and U. S. R. Murty. *Graph theory with applications*, volume 290. Macmillan, London, 1976.

[3] I.M. Branco and J. D. Coelho. The hamiltonian p-median problem. *European Journal of Operational Research*, 47:86–95, 1990.

[4] H. Broersma and X. Li. Spanning trees with many or few colors in edge-colored graphs. *Discussiones Mathematicae Graph Theory*, 17:259–269, 1997.

[5] H. Broersma and X. Li. Spanning trees with many or few colors in edge-colored graphs. *Graph Theory*, 17:259–269, 1997.

[6] F. Carrabs, R. Cerulli, M. Gaudioso, and M. Gentili. Lower and upper bounds for the spanning tree with minimum branch vertices. *Computational Optimization and Applications*, 56:405–438, 2013.

[7] C. Cerrone, R. Cerulli, and A. Raiconi. Relations, models and a memetic approach for three degree-dependent spanning tree problems. *European Journal of Operational Research*, 232:442–453, 2014.

[8] R. Cerulli, P. Dell'Olmo, M. Gentili, and A. Raiconi. Heuristic approaches for the minimum labelling hamiltonian cycle problem. *Electronic notes in Discrete Mathematics*, 25:131–138, 2006.

[9] R. Cerulli, A. Fink, M. Gentili, and A. Raiconi. The k-labeled spanning forest problem. *Procedia-Social and Behavioral Sciences*, 108:153–163, 2014.

[10] R. Cerulli, M. Gentili, and A. Iossa. Bounded-degree spanning tree problems: models and new algorithms. *Computational Optimization and Applications*, 42:353–370, 2009.

[11] R.S. Chang and S.J. Leu. The minimum labeling spanning trees. *Information Processing Letters*, 63:277–282, 1997.

[12] M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming.* Springer, Berlin, 2014.

[13] A. S. da Cunha, L. Simonetti, A. Lucena, and B. Gendron. Formulations and exact solution approaches for the degree preserving spanning tree problem. *Networks*, 2015.

[14] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8:250–255, 1965.

[15] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2:393–410, 1954.

[16] J. Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial Structures and Their Applications*, pages 69–87, 1970.

[17] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 8:128–140, 1736.

[18] M. Fischetti, J.-J. Salazar González, and P. Toth. Experiments with a multicommodity formulation for the symmetric capacitated vehicle routing problem. In *Proceedings of the 3rd Meeting of the EURO Working Group on Transportation*, pages 169–173, 1995.

[19] T. Fujie. The maximum-leaf spanning tree problem: Formulations and facets. *Networks*, 43:212–223, 2004.

[20] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32:835–859, 1977.

[21] M. R. Garey and D. S. Johnson. Computers and Intractability: A guide to the theory of NP-Completeness. *W. H. Freeman, New York*, 1979.

[22] L. Gargano, P. Hell, L. Stacho, and U. Vaccaro. Spanning trees with bounded number of branch vertices. In *Automata, Languages and Programming*, pages 355–365. Springer Berlin Heidelberg, 2002.

[23] E.N. Gilbert and H.O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16:1–29, 1968.

[24] R. E. Gomory. An algorithm for integer solutions to linear programs. *Recent Advances in Mathematical Programming*, 64:269–302, 1963.

[25] I. Gribkovskaia, Ø. Halskau, and G. Laporte. The bridges of Königsberg—a historical perspective. *Networks*, 49:199–203, 2007.

[26] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* North-Holland, Amsterdam, 1992.

[27] N. Jozefowiez, G. Laporte, and F. Semet. A branch-and-cut algorithm for the minimum labeling hamiltonian cycle problem and two variants. *Computers & Operations Research*, 38:1534–1542, 2011.

[28] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 225–330. Elsevier, North-Holland, Amsterdam, 1995.

[29] S. Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM Journal on Computing*, 24:247–265, 1995.

[30] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[31] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[32] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:231–247, 1992.

[33] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[34] X. Li and X.Y. Zhang. On the minimum monochromatic or multicolored subgraph partition problems. *Theoretical Computer Science*, 385:1–10, 2007.

[35] A. Lucena, N. Maculan, and L. Simonetti. Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7:289–311, 2010.

[36] T. L. Magnanti and L. A. Wolsey. Optimal trees. In M. O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Network Models, Handbooks in Operations Research and Management Science 6*, pages 503–615. North-Holland, Amsterdam, 1995.

[37] A. Marín. Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem. *European Journal of Operational Research*, 245:680–689, 2015.

[38] R. A. Melo, P. Samer, and S. Urrutia. An effective decomposition approach and heuristics to generate spanning trees with a small number of branch vertices. *arXiv preprint arXiv:1509.06562*, 2015.

[39] M. Merabet, S. Durand, and M. Molnar. Minimization of branching in the optical trees with constraints on the degree of nodes. In *ICN'12: The Eleventh International Conference on Networks*, pages 235–240, Saint-Gilles, Réunion Island, 2012.

[40] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the Association for Computing Machinery*, 7:326–329, 1960.

[41] J. E. Mitchell. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of Applied Optimization*, pages 65–77, 2002.

[42] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, New York, 2014.

[43] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:60–100, 1991.

[44] M. W. Padberg and L. A. Wolsey. Trees and cuts. In *Combinatorial Mathematics, Annals of Discrete Mathematics 17*, pages 511–517. North-Holland, Amsterdam, 1983.

[45] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[46] J.-J. Salazar-Gonzalez. The steiner cycle polytope. *European Journal of Operational Research*, 147:671–679, 2003.

[47] J. M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Information Processing Letters*, 113:241–244, 2013.

[48] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, 1998.

[49] D. M. Silva, R. M. A. Silva, G. R. Mateus, J. F. Gonçalves, M. G. C. Resende, and P. Festa. An iterative refinement algorithm for the minimum branch vertices problem. In *Experimental Algorithms*, pages 421–433. Springer, Berlin Heidelberg, 2011.

[50] R. M. A. Silva, D. M. Silva, M. G. C. Resende, G. R. Mateus, J. F. Gonçalves, and P. Festa. An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. *Optimization Letters*, 8:1225–1243, 2014.

[51] S. Silvestri, G. Laporte, and R. Cerulli. The rainbow cycle colver problem. Technical Report CIRRELT-2015-40, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT), August 2015.

[52] S. Sundar, A. Singh, and A. Rossi. New heuristics for two bounded-degree spanning tree problems. *Information Sciences*, 195:226–240, 2012.

[53] J. W Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14:325–336, 1984.

[54] J.W. Suurballe. Disjoint paths in a network. *Networks*, 4:125–145, 1974.

[55] L. A. Wolsey. *Integer Programming.* Wiley, Hoboken, New Jersey, 1998.

[56] Y. Xiong, B.L. Golden, and E.A. Wasil. The colorful traveling salesman problem. In *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies*, pages 115–123. Boston: Springer, 2007.