UNIVERSITY OF SALERNO
Departments of Mathematics and Physics

PhD in Mathematics, Physics and Applications
Curriculum: Mathematics
CYCLE XXX

Conflicting edges spanning trees and NP-Hard subgraph identification problems

Tutor:
Prof. Raffaele Cerulli

Doctoral dissertation of :
Rosa Pentangelo

Co-Tutor:
Prof. Francesco Carrabs

Coordinator:
Prof. Roberto Scarpa

2016/2017

A te che sei l'amore piú immenso e puro...

Acknowledgements

I thank my tutor with great esteem, prof. Raffaele Cerulli, who was able to help me and direct me towards the development of my work throughout my career. I also thank the prof. Francesco Carrabs, co-tutor of this thesis, who with his professionalism has always been available to help me and support me.

I thank all the members of my group at the University of Salerno with deep affection, with whom I shared day-by-day satisfactions and disappointments. They have been fundamental to the success of my studies. Thanks to Ciro, Federica, Andrea and Carmine.

A huge thank you goes to my biggest supporter, my husband Damiano. He always knows how to spur me, reminding me every time how proud he is of me.

A special thanks to my immense family, in particular to my great brother Alessandro, to my uncles and cousins who have always supported me with great confidence. Together with them, I thank my in-laws and Donatella. The greatest thank you goes to the woman of my life, my immense grandmother Regina who never allowed me to give up; several times I think I owe her all the results achieved.

Thanks to my magical granddaughter Aurora, who allowed me to overcome the difficulties encountered during this journey, sometimes just holding her in my arms. Thanks to Caterina, Ciro, Carolina and Enzo for always been there.

Thanks to my colleagues and friends Arianna, Rosaria and Valentina, with whom, despite the passing of the years and the different paths chosen, we continue to support each other.

Ringrazio con profonda stima il mio tutor, il prof. Raffaele Cerulli, che ha saputo aiutarmi e indirizzarmi verso lo svolgimento del lavoro durante tutto il mio percorso. Inoltre ringrazio il prof. Francesco Carrabs, co-tutor di questa tesi, che con la sua professionalità si è reso sempre disponibile nell' aiutarmi e sostenermi.

Ringrazio con profondo affetto tutti i componenti del mio gruppo dell' Università di Salerno, con loro ho condiviso giorno per giorno soddisfazioni e delusioni. Sono stati fondamentali per la riuscita del mio percorso. Grazie a Ciro, Federica, Andrea e Carmine.

Un immenso grazie va al mio più grande sostenitore, mio marito Damiano. Lui che sa sempre come spronarmi, ricordandomi ogni volta quanto è orgoglioso di me.

Un grazie speciale alla mia immensa famiglia, in particolare al mio grande fratello Alessandro, ai miei zii e cugini che mi hanno sempre sostenuta con estrema fiducia. Insieme a loro, ringrazio i miei suoceri e Donatella. Il grazie più grande va alla donna della mia vita, la mia immensa nonna Regina che non mi ha mai permesso di arrendermi; più volte penso che devo a lei tutti i risultati raggiunti.

Grazie alla mia magica nipotina Aurora, che mi ha permesso di superare le difficoltà incontrate durante questo percorso, talvolta solo tenendola tra le mie braccia. Grazie a Caterina, Ciro, Carolina ed Enzo per esserci sempre stati.

Grazie alle mie colleghe e amiche Arianna, Rosaria e Valentina, con le quali nonostante il passare degli anni e i differenti percorsi scelti, continuiamo a sostenerci a vicenda.

Contents

Contents vi

List of Figures x

1 Some important concepts in the integer and combinatorial optimization 16
- 1.1 Graph theory introduction . . . 17
- 1.2 Polyhedral theory . . . 18

2 Subgraph identification problems 22
- 2.1 The Spanning Tree problems . . . 22
  - 2.1.1 *NP*-hard variants of minimum spanning tree problem . . . 25
- 2.2 Arc routing problems . . . 27
  - 2.2.1 The Rural Postman Problem . . . 27
- 2.3 Shortest path problems . . . 29
  - 2.3.1 All Color Shortest Path problem . . . 30

3 Minimum spanning tree problem with conflicting edge pairs 32
- 3.1 Introduction . . . 32
- 3.2 Notations and problem definition . . . 33
- 3.3 A Multi-Ethnic Genetic Approach for the Minimum Conflict Weighted Spanning Tree Problem . . . 34
  - 3.3.1 The Genetic Algorithm . . . 35
  - 3.3.2 Improvement procedures . . . 39
  - 3.3.3 Multi Ethnic Genetic Approach . . . 41
- 3.4 A Branch-and-Cut approach for the MSTC problem . . . 44

3.4.1 Basic Mathematical Model . . . 45
3.4.2 Valid inequalities . . . 45
3.4.3 Branch-and-Cut approach . . . 49
3.5 Computational results . . . 50
3.5.1 Mega . . . 51
3.5.2 Branch-and-Cut approach . . . 57

4 A flow formulation for the Close-enough arc routing problem 72
4.1 Introduction . . . 72
4.2 Problem definition . . . 72
4.3 Flow formulation . . . 73
4.3.1 Graph reduction . . . 73
4.3.2 The Vertex Cover . . . 74
4.3.3 MIP model . . . 75
4.4 Computational results . . . 78

5 Properties, formulation and a two-level metaheuristic for the all-colors shortest path problem 80
5.1 Introduction . . . 80
5.2 Problem definition and properties . . . 82
5.3 Mathematical model . . . 87
5.3.1 Valid inequalities . . . 89
5.3.2 Adapting the model to ACSP-SC and ACSP-SV . . . 90
5.4 Variable Neighborhood Search . . . 91
5.4.1 Initialization algorithm . . . 92
5.4.2 Relocate and 2-Opt Local Search . . . 93
5.4.3 Shake operator . . . 98
5.5 Computational results . . . 98
5.5.1 Comparisons on the ACSP-SV problem . . . 98
5.5.2 Comparisons on the ACSP problem . . . 103

6 Conclusions 108
6.1 The Minimum Spanning Tree problem with Conflicting Edge Pairs (Chapter 3) . . . 108

6.2 The Close-Enough Arc Routing problem (Chapter 4) 109
6.3 The All Color Shortest Path problem (Chapter 5) 109

References **112**

List of Figures

1.1 Black dots to represent the integer points in S. The shaded area is the convex hull of S. 20

2.1 Schematic representation of the Königsber (XVIII century), currently Kaliningrad. 28

3.1 (a) A generic graph G. (b) A spanning tree T_1 of G with $W(T_1) = 23$. (c) A spanning tree T_2 of G with $W(T_2) = 20$ 34

3.2 (a) A graph G. (b) A spanning tree T_1 of G. (c) The chromosome representing T_1. 36

3.3 (a) Two spanning trees of G. (b) The subgraph G' generated by crossover. (c) The new child chromosome T_c. 38

3.4 The improvement procedures framework applied on each chromosome of the final population. 42

3.5 Multi ethnic genetic framework 43

3.6 (a) An example of graph G with $|V| = 6$, $|E| = 9$ and conflicts set $P = \{\{e_1,e_5\},\{e_1,e_7\},\{e_1,e_9\},\{e_3,e_4\},\{e_3,e_7\},\{e_3,e_9\},\{e_4,e_7\}\}$ ($|P| = 7$). (b) The related conflict graph $G' = (E,P)$, where each node corresponds to an edge of G and each edge corresponds to a pair in P. . . . 44

3.7 (a) The input graph G. (b) A feasible solution that satisfies constraints (2)-(4) and the inequalities $0 \leq x_e \leq 1, \forall e \in E$. (c) Considering the cycle $\Psi = \{e_4,e_5,e_9,e_8\}$ in the solution (b) and edge e_1 in conflict with e_5 and e_9, the related constraint (7) is violated. 48

3.8 (a) An odd-cycle of length 5 in the conflict graph G' in Figure 3.6. (b) If we choose e_1, it is not possible to choose e_5 and e_9. (c) At this point, only one between e_3 and e_4 can be part of a MSTC solution. 48
3.9 Performance comparison of TS vs Mega from the results of Table 3.4. 57

4.1 Redundant target m_2. Necessary vertex A. Necessary arc (E,F). . . . 75
4.2 (a) Edges in which there is at least one target (red dot). (b) All edges in which it is possible to read at least one target. (c) A feasible solution. 76

5.1 Example graph with 8 vertices, 12 edges and 7 colors 81
5.2 Low-level and high-level optimal solution for the instance of Figure 5.1 86
5.3 Neighbor construction through Relocate Local Search (step 1, $i = 4$) . 96
5.4 Neighbor construction through Relocate Local Search (step 2, $j = 2$) . 97
5.5 Neighbor construction through 2-opt Local Search ($i = 3$, $j = 5$) . . . 105

Introduction

How often do we try to get the best result with the least effort, spend as little time as possible to perform a task or make the most of the resources available in the workplace? In everyday life, the word ”optimize” is therefore often present. In particular, the optimization has as its object the study and the development of quantitative methodologies and tools for the solution of decision problems. This is a discipline born in the military field about 80 years ago. Over the years, it has found application in several sectors such as logistics and production, finance and telecommunications. Currently it has become an indispensable tool for supporting decision-making processes. The problems faced are typically those in which decisions have to be made on the use of resources available in limited quantities in order to respect an assigned set of constraints, maximizing, for example, the benefit obtainable from the use of the resources themselves.

Many problems of real-life as also problems of theoretical importance in the field of operational research are combinatorial in nature. Combinatorial Optimization studies the optimization problems in which the feasible set is defined in terms of combinatorial structures. Many of these problems are defined on graphs (directed or undirected). The fundamental characteristic of these problems is therefore to be defined on discrete space.

This doctoral thesis involves the study of three different combinatorial optimization problems defined on graphs: the *Minimum Spanning Tree problem with Conflicting Edge Pairs* (MSTC), the *Close-enough Arc Routing problem* (CEARP) and the *All Color Shortest Path problem* (ACSP). All

these problems concern the identification of subgraphs and they are variants of well known problems. In the following the definition of these problems are reported.

- MSTC: Let $G(V,E,P)$ be an undirected edge weighted graph, where V is the set of n vertices, E the set of m edges and $P \subseteq E \times E$ is the set of *conflict edge pairs*. The MSTC consists of finding a minimum spanning tree of G without conflicting edge pairs, that is for each pair $\{e_i, e_j\} \in P$ at most one between e_i and e_j belongs to the edges of the spanning tree. The MSTC has been shown to be *NP*-hard.

- CEARP: Let $G = (V,A,M)$ be a directed graph with a set of vertices V, a set of arcs A, and a set of targets M located on arcs. Each arc in A has a cost and an arc $a \in A$ covers a target $m \in M$ iff the target is either on the arc or within a predetermined distance (radius) from the arc. Fixed a depot node, the *CEARP* consists of finding a minimum cost tour starting and ending at the depot node, traversing a subset of arcs such that all the targets in M are covered. The CEARP has been shown to be *NP*-hard.

- ACSP: Let $G = (V,E,C)$ be an undirected, connected and vertex labeled graph with V the set of vertices, E the set of the edges and C the set of labels (or colors). The aim of ACSP is to find a path of minimum cost such that all colors are reached at least once. The ACSP has been shown to be *NP*-hard.

This dissertation is organized as follows. Chapter 1 provides some basic concepts and definitions needed to understand subsequent content. We will analyze some elementary definitions of *graph theory*, as well as some concepts of *polyhedral theory*.
Chapter 2 describes the *Minimum Spanning Tree problem, the Arc Routing problem and the Shortest Path problem* from which MSTC, CEARP and

ACSP are originated, respectively. These last problems are addressed in Chapter 3, 4 and 5. In Chapter 3 we propose a multi ethnic genetic algorithm for the MSTC problem. Moreover three local search procedures are developed to improve the solutions inside the population during the computation [9]. Furthermore, we introduce a new set of valid inequalities for the problem, based on the properties of its feasible solutions, and we develop a Branch-and-Cut algorithm based on them [11]. Computational tests are carried out on the benchmark instances proposed in literature and on a new set of randomly generated instances.

In Chapter 4 we face the CEARP problem and we propose some techniques to reduce the size of the input graph and a new effective mixed integer programming (MIP) formulation for the problem [15]. The effectiveness of the reduction techniques are showed. Computational results obtained by comparing our MIP model with the existing exact methods show that our algorithm is really effective in practice.

In Chapter 5 we propose some new properties for the ACSP problem, as well as a compact representation of the feasible solutions. Furthermore, we present a novel mathematical formulation and a metaheuristic approach based on these ideas [10]. Computational results show the effectiveness of our approach with respect to previous contributions proposed in literature.

A summary of the obtained results are reported at the end of this dissertation. Finally we present some remarks and future research directions.

Chapter 1

Some important concepts in the integer and combinatorial optimization

Many real-life problems can be faced by using mathematical programming models (with variables of integer or continuous type). In particular, this approach plays an important role in the resolution of real-life problems for which there is something to be maximized or minimized, having to take into account some constraints and restrictions. It is a matter of considering a function of several variables (*objective function*) to be maximized or minimized, subject to a set of constraints that constitute the *set of feasible solutions*.

It would be enough to observe our daily life to realize the problems that can be translated into optimization problems, such as determining the least long or the least expensive route to go to work, manage the budget for the grocery shopping spending the least time, etc. As it seems obvious that to solve this type of problems (when the dimension is small) is not essential to use specific tools, however, they are essential in the resolution of more complex problems. The aim therefore is to solve optimization problems thanks to the help of the well-known tools in order to obtain the best possible solution, when it is possible. We call *optimal solution* the best possible solution, i.e. the feasible solution for which the objective function assume the minimum (or maxi-

mum) value.

In this chapter we introduce some basic concepts and definitions used in the next chapters. In particular we give some elementary concepts of graph and polyhedral theory.
The concepts described in this chapter refer to books [51], [60], [65] and [35].

1.1 Graph theory introduction

Graph Theory (introduced by the Swiss mathematician Euler (1707 - 1783)) constitutes, like Mathematical Programming, a methodological body for modeling and solving decision problems. Among all, the problems we are going to face in these next chapters concern the identification of particular subgraphs. It is therefore necessary to give a brief overview of the basic concepts of graph theory.

A *graph* G consists of a sets couple (V, E) in which V is called the set of nodes and E is called the set of edges. E is a subset of all the possible pairs of nodes in V. We assume that V has cardinality n and E has cardinality m. If the pairs of nodes are ordered, the graph is said *directed*, if are not ordered, the graph is said *undirected*. From now on, when we refer to a directed graph we will talk about arcs and not edges and we will indicate the graph with the pair of sets (V, A). Two edges that have a common node are called *adjacent*. Furthermore, if there exists an edge $e = (u,v) \in E$, we say that the nodes v and u are adjacent, they are the endpoints of e and e is incident to v and u. Following these definitions, we consider the set of all edges incident to a node v, denoted by $\delta(v)$, so we can define the *degree* of the node v as the number of these incident edges, i.e. $|\delta(v)|$. When the degree of node v is equal to zero, i.e. there are not incident edges to v, the node is called *isolated*. Given a set $S \subseteq V$, with E(S) we indicate the set of edges with both endpoints in S.
A sequence of $k+1$ nodes $v_0, v_1, ..., v_k$ such that, for every $i = 1, ..., k$, $(v_{i-1}, v_i) \in E$ or $(v_i, v_{i-1}) \in E$ is called *path* (directed path if the graph is directed) and its length is k. We can also see a path of length k as a sequence of adjacent edges two by two. A path is called *simple* if no node is crossed more than once. A simple path in which the first and last nodes coincide (i.e. $v_0 = v_k$) is called *cycle* of length k.

A graph is *complete* if there is an edge between any pair of nodes. Instead, a *connected* graph is a graph that contains a single *connected component* (a subgraph where each node is connected to all other nodes through a path). Considering a graph $G = (V,E)$, a subgraph $G^- = (V^-,E^-)$ of G is a graph in which $V^- \subseteq V$ and $E^- \subseteq E$. Given a graph $G = (V,E)$, we say *Eulerian cycle* in G a cycle in G which crosses exactly once all the edges of G. A graph $G = (V,E)$ is called *Eulerian graph* if it contains at least one *Eulerian* cycle.

1.2 Polyhedral theory

In this section we provide some basic results from linear algebra and some results regarding the polyhedral theory. In the integer programming, one of the principal objectives is finding a linear inequality description of the set of feasible points. Let $\mathbb{R}^n$ be the set of n-dimensional vectors.

Definition 1.2.1 *A set of points $x_1,...,x_k \in \mathbb{R}^n$ is linear independent if the unique solution of $\sum_{i=1}^k \lambda_i x_i = 0$ is $\lambda_i = 0$, $i = 1,...,k$, where $\lambda_i \in \mathbb{R}$.*

The maximum number of linearly independent points in $\mathbb{R}^n$ is n.

Definition 1.2.2 *A set of points $x_1,...,x_k \in \mathbb{R}^n$ is affinely independent if the unique solution of $\sum_{i=1}^k \alpha_i x_i = 0$, $\sum_{i=1}^k \alpha_i = 0$ is $\alpha_i = 0$, $i = 1,...,k$, where $\alpha_i \in \mathbb{R}$.*

Note that the linear independence implies the affine independent, but the affine independence does not implies the linear independence. To clarify the relation between them we can observe the following proposition.

Proposition 1.2.1 *The following statements are equivalent:*

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

The maximum number of affinely independent points in $\mathbb{R}^n$ is $n+1$, just consider n linearly independent points in $\mathbb{R}^n$ and the zero vector.

Definition 1.2.3 *A polyhedron $P \subseteq \mathbb{R}^n$ is the set of points that satisfy a finite set of linear inequalities, i.e. $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, where $A \in \mathbb{R}^{h \times n}$ and $b \in \mathbb{R}^h$.*

A bounded polyhedron is called polytope.

Definition 1.2.4 *A polyhedron P is of dimension k ($dim(P) = k$) if the maximum number of affinely independent points in P is $k+1$.*

Definition 1.2.5 *A polyhedron $P \subseteq \mathbb{R}^n$ is full-dimensional if $dim(P) = n$.*

Following these definitions, we will see that surely a polyhedron is not full-dimensional if at least one of the inequalities that define it, is satisfied with the equality from all its points. To this aim we consider the sets $M = \{1, ..., m\}$, $M^{=} = \{i \in M : a^i x = b_i$ for all $x \in P\}$ and $M^{\leq} = \{i \in M : a^i x < b_i$ for some $x \in P\}$. Note that $M^{\leq} = M \setminus M^{=}$. In fact, let $(A^{=}, b^{=})$ and $(A^{\leq}, b^{\leq})$ be corresponding rows of (A, b), the following result is valid.

Proposition 1.2.2 *If $P \subseteq \mathbb{R}^n$, then dim(P)+ rank$(A^{=}, b^{=})$= n*

Given a polyhedron $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, there are some inequalities that are necessary in describing P, while some others can be eliminated.

Definition 1.2.6 *Given an inequality $\pi x \leq \pi_0$, we say that $\pi x \leq \pi_0$ is a valid inequality for P if it is satisfy by all points in P.*

Definition 1.2.7 *If $\pi x \leq \pi_0$ is a valid inequality for P and if $F = \{x \in P : \pi x = \pi_0\}$, F is called face of P. Moreover, if $F \neq \emptyset$ and $F \neq P$, then F is a proper face induced by $\pi x \leq \pi_0$.*

Definition 1.2.8 *A face F of P is a facet of P if $dim(F) = dim(P) - 1$.*

In the description of a polyhedron P, the facets are necessary and sufficient for the description of P.

If P is full-dimensional, P possesses a single *minimal description* so that in the linear system that represents it, each inequality is unique to less than a positive multiplier.

In general, in a minimal description, each inequality or equality is necessary, therefore removing even one, the polyhedron P changes.

To the aim of finding a linear inequality description of the set of feasible points, in the integer programming, we can give some definitions:

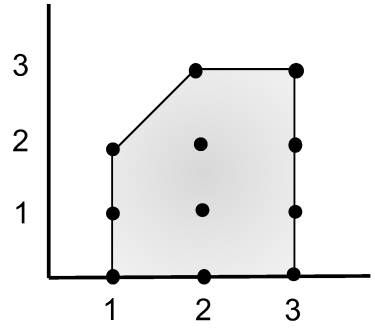


Figure 1.1: Black dots to represent the integer points in S. The shaded area is the convex hull of S.

Definition 1.2.9 *Given a set $S \subseteq \mathbb{R}^n$, a point $x \in \mathbb{R}^n$ is a convex combination of points of S if there exists a finte set of points $x_1, \ldots, x_t \in S$ and $\lambda_1, \ldots, \lambda_t \in \mathbb{R}^+$ with $\sum_{i=1}^{t} \lambda_i = 1$ and $x = \sum_{i=1}^{t} \lambda_i x_i$.*

Definition 1.2.10 *Given a set $S \subseteq \mathbb{R}^n$, the convex hull of S is the set of all points that are convex combinations of points in S. The convex hull of S is denoted by $conv(S)$.*

Figure 1.1 shows the convex hull of a set of integers point in $\mathbb{R}^2$. Note that, if we have 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 $conv(S)$ is very important.

Chapter 2

Subgraph identification problems

In this thesis we will face *NP*-hard variants of three well-known and studied problems in the context of combinatorial optimization. The purpose of this chapter is to describe these variants and the problems from which they originate: the *Minimum Spanning Tree problem, the Arc Routing problem and the Shortest Path problem*. Each of these problems can be seen as a subgraph identification problems.

2.1 The Spanning Tree problems

The problem of finding a minimum spanning tree (MST) of a weighted undirected graph is one of the best studied problems in the area of combinatorial optimization. In this section we present the problem and two well-known mathematical formulations for this problem. Then we explain some related problems and a specific *NP*-hard variant of the MST problem named "Minimum Spanning Tree problem with Conflicting Edge Pairs".

Let $G = (V, E)$ be an undirected connected graph, where V is the set of the vertices and E is the set of the edges, respectively with cardinality n and m. A spanning tree $T = (V, E_T)$ of G is a connected and acyclic subgraph containing all vertices and a subset $E_T \subseteq E$ of the edges in G, that is a tree that spans over all vertices in G. Any spanning tree of a connected graph with n vertices has exactly $n-1$ edges. If we consider a connected weighted graph $G = (V, E)$ with $w : E \longrightarrow R^+$ function that assigns a

weight w_e to each edge $e \in E$, the MST problem consists of finding a spanning tree $T = (V, E_T)$ with the minimum weight. The weight of a tree $T(V, E_T)$ is the sum of the edge weights in E_T:

$$w(T) = \sum_{e \in E_T} w_e. \tag{2.1}$$

The MST problem has direct applications in networks design context, including computer networks, telecommunications networks, transportation networks, water supply networks, and electrical grids.

A classical application of the MST regards the problem a telecommunications company must resolve laying cable to in new neighborhood. If the company is constrained to bury the cable only along certain paths (e.g. along roads), then you can consider a graph representing which points are connected by those paths. Some of those paths might be more expensive because, for example, they are longer, or require the cable to be buried deeper; these paths would be represented by edges with larger weights. A spanning tree for that graph would be a subset of those paths that has no cycles but still connects to every house; there might be several spanning trees possible. A minimum spanning tree would be one with the lowest total cost, thus would represent the least expensive path for laying the cable.

The MST problem can be formulated as follow, using decision variables x_e associated with the edges of the graph G, with the following meaning:

$$x_e = \begin{cases} 1 & \text{if the edge } e \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$$

Using the classical subtour elimination constraints, an integer mathematical programming formulation for MST is the following:

$$\min \sum_{e \in E} w_e x_e \tag{2.2}$$

subject to

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

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \qquad \forall S \subseteq V, |S| \geq 3 \tag{2.4}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{2.5}$$

In this model, the objective function (2.2) minimizes the total weight of the spanning tree. Constraint (2.3) indicates that exactly the $n-1$ edges (where $|V| = n$) can be chosen in the solution while Constraints (2.4) are the classical subtour elimination constraints to respect the request to avoid cycle in the solution. Finally, Constraints (2.5) are variable definitions.

Integer programming problems are, in general, difficult to solve. Often a relaxation of the original problem, more easy to solve, is solved. Among all the types of relaxation, the best known is the linear relaxation obtained by eliminating the restriction that the decisional variables x_e in the model (2.2)-(2.5) must be integers. In general, by eliminating the integrality constraint from the model, the objective function (min) of linear programming relaxation will have a lower optimal value than the integer programming; for the minimum spanning tree problem, this does not happen. In particular, the integer programming formulation and the linear programming formulation have the same optimal value. This property is summarized in the follow result by Edmonds in [23].

Theorem 2.1.1 *The extreme points of the polyhedron defined by the linear programming relaxation of the spanning tree model (2.3)- (2.5) are the 0-1 incidence vectors of spanning trees.*

In formulating the spanning tree problem as an integer program, we used the property that a spanning tree of a graph $G = (V,E)$ with n nodes is any subgraph containing $n-1$ edges that do not form cycles. Therefore, at most $|S|-1$ edges in any tree can connect any subset S of nodes in V. As an alternative, we could use a different but equivalent definition of a spanning tree: it is a connected subgraph containing $n-1$

edges. This definition leads to the following *Cutset formulation*:

$$\min \sum_{e \in E} w_e x_e \tag{2.6}$$

subject to

$$\sum_{e \in E} x_e = |V| - 1 \tag{2.7}$$

$$\sum_{e \in \delta(S)} x_e \geq 1, \qquad \forall S \subset V, S \neq \emptyset \tag{2.8}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{2.9}$$

The cutset $\delta(S)$ is a subset of edges with one end in S and the other end in $V \setminus S$. Constraints (2.8) ensure that subsets S and $V \setminus S$ are connected.

2.1.1 *NP*-hard variants of minimum spanning tree problem

Many real problems can be modeled by graphs for which a spanning tree is required as solution. Obviously in the real word it is necessary to take into account the different constraints and specific characteristics of the problem to be solved; this have given rise to the definition of more variants of spanning tree problems. Among all, in the next subsections, we describe some it NP-hard variants like the Generalized Minimum Spanning tree [50], the Spanning Tree with Minimum Branch Vertices [32] and the Minimum Spanning Tree problem with Conflicting Edge Pairs ([17], [18]).

The Generalized Minimum Spanning Tree problem Let $G = (V,E)$ be an undirected graph with n nodes in V and m edges in E. For each edge $e_i \in E$ there is an associate weight w_{e_i}. The set V is partitioned in $k \in K$ clusters V_k and the set $E = \{\{i,j\} : i \in V_{k_1}, j \in V_{k_2} \quad s.t. \quad k_1 \neq k_2\}$. To solve the generalized minimum spanning tree problem (GMSTP) it is necessary to determine a tree of minimum total cost that include one node for each cluster V_k. This problem was first introduced by [50]

and the same authors proved that GMSTP is NP-Hard. An application in the real world of GMSTP ([50]) is identified in the problem to choose how to position regional public structures or service centers, which need to be connected to each other. In particular, we are facing a GMSTP even when planning metropolitan networks and road networks.

The Minimum Branch Vertices problem Given a connected graph $G = (V, E)$ with the set V of vertices and the set E of the edges, the spanning tree with the minimum branch vertices is the spanning tree with the minimum number of vertices with degree greater than two. The interest in this problem arises for applications in optical networks. In this context, it is necessary to connect the number of nodes limiting the number of connections of each node. In particular, for this problem, the interest borns to the practical application in the context of all-optical networks which are a class of backbone wide area networks (WAN) where connections are routed by intermediate nodes in the optical domain without electronic conversion. More details regarding this application are reported in [7]. The problem was presented by Gargano et al. [32]. In the same paper, the authors analyzed computational complexity and proved it to be NP-hard.

The Minimum Spanning Tree problem with Conflicting Edge Pairs The Minimum Spanning Tree Problem with Conflicting Edge Pairs (MSTC) is a NP-Hard variant of the classical Minimum Spanning Tree problem. Let $G(V, E, P)$ be an undirected and edge weighted graph, P is a set of conflicting edge pairs so the MSTC problem consists of finding a spanning tree of G of minimum cost without edges in conflict, i.e. for each couple of edges $e_i, e_j \in P$, a solution of the MSTC can contain at most one between e_i and e_j. The problem was introduced by Darmann et al. [17], [18] and arises in some real world applications like the installation of an oil pipeline system connecting various countries [17] or the search of paths in a map of the city with restrictions [43]. This problem also finds application in the design of an offshore wind farm network. The design of such systems is based on the connection layout of wind turbines installed, realized through cables characterized by a certain capacity and a

certain cost. Given the capacity of each cable, the system design begins by performing a clustering of the turbines that can be connected to a single cable. Defined a cluster of turbines, the next step consists of connecting them in the cheapest way (thus creating a spanning tree of minimum cost). The additional request is to carry out this connection by avoiding overlapped cables ([44]). By considering two overlapped cables as a conflicting pair, the problem just described coincides with the MSTC. Other applications are mentioned in [66].

2.2 Arc routing problems

The purpose of arc routing problems is to determine a minimum cost route of a specified subset of a graph. Given the presence of problems of this kind in many real fields, they have long been the subject of attention by mathematicians and operations researchers. Already since 1736, the Swiss mathematician Euler faced the first documented problem of arc routing ([26]): the famous problem of the bridge of Königsberg. It was a matter of proving the existence of a closed walk that crossed exactly once each of the seven bridges over the Pregel river in Königsberg (Figure 2.1). Among the arc routing problems, we also find the problem of the *Chinese postman* formulated by the mathematician Meigu Gaun ([36]) who, during the Chinese cultural revolution, worked as a post office worker. The problem can be formulated as follows: suppose there is a postman who must deliver mail to a particular neighborhood. The postman wants to find the shortest route in the neighborhood starting and ending in the same point, covering each street at least once. For an Eulerian graph, an Eulerian cycle is the optimal solution. Other real contexts in which we find arc routing problems are, for example, waste collection, snow removal and school bus routing. These problems have been addressed by operational researchers after that for years of countless sums have been wasted for these operations.

2.2.1 The Rural Postman Problem

Given a directed graph $G = (V,A)$, in the Chinese postman problem, we look for a closed walk that covers all the arcs in the graph with the minimum cost. In most real contexts, however, the request is to cover a part of the arcs of a graph but not all. In

these cases, in which the path must involve a subset $R \subseteq A$ of arcs of G, the problem becomes a *Rural Postman problem (RP)*. In [47], Lenstra and Rinnoy Kan showed that RP is NP-hard. More details about these problems are presented in [24] and [25].

The Close-Enough Arc routing Problem The *Close-Enough Arc Routing Problem (CEARP)* is a generalization of the Rural Postman problem in which we consider a directed graph $G = (V,A,M)$ with a set of vertices V, a set of arcs A, and a set of targets M located on arcs. Traversing an arc have a cost; an arc $a \in A$ covers a target $m \in M$ iff the target is either on the arc or within a predetermined distance (radius) from the arc. Fixing a depot node, the *CEARP* consists of finding a minimum cost tour starting and ending at the depot node, traversing a subset of arcs such that all the targets in M are covered. The CEARP problem was introduced by Drexl [20, 21] that proved the problem is NP-hard, and he proposed a branch-and-cut algorithm to solve it. There are several real-life applications for this problem. The meter reading problem is an important application of the CEARP: A vehicle with a receiver on board travels over a street network. If it traverses a street and is closer than a certain distance to a

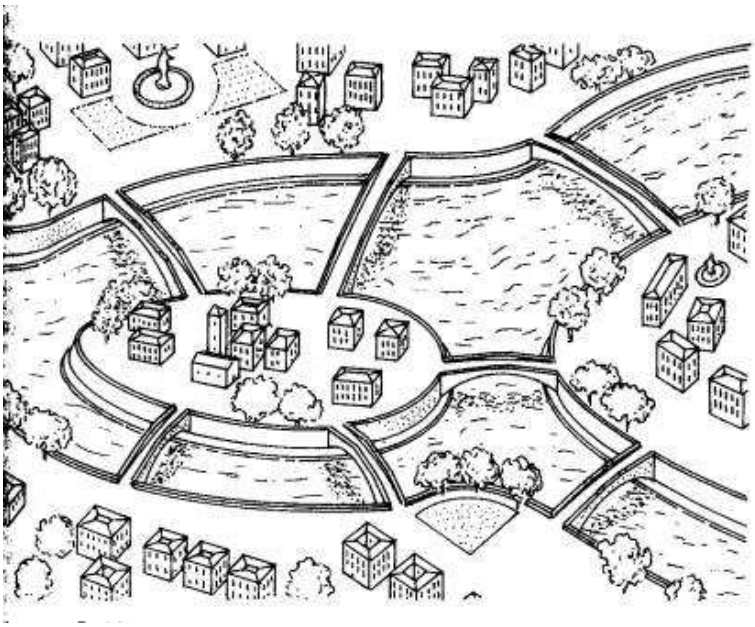


Figure 2.1: Schematic representation of the Königsber (XVIII century), currently Kaliningrad.

RFID meter (e.g., in a home), the receiver is able to read the value of the meter. An interesting variant of this real-life problem is when flying drones are used to read the meters [8].

2.3 Shortest path problems

When we have to find a path to go from a location to another, it would be ideal to be able to choose the path that requires the least cost. In reality, to realize the least expensive route, one must consider the different arcs cost to travel between a certain number of intermediate points. In these cases we are dealing with shortest path problems. The shortest path problems are problems in which we want to find a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent arcs is minimized. In same cases "shortest path" is not related to the distance between one place and another but it could be referred to the shortest route in terms of time or to the less expensive route in terms of monetary cost.

Shortest path problems are closely related to our daily life. For example, when we are traveling to get to work location, we should choose our route so that we can travel less time to reach the destination. To this end, we could evaluate as the cost of each intermediate path the time to be used for this segment, calculating, for example, this cost in terms of vehicular traffic or uneven road. However, in the real life situations, there are many possible routes and it is quite impossible for us to try all the possible routes to find the one with the shortest amount of time. Therefore, we need a more effective method to discover this path.

In a shortest path problem, given a directed or undirected weighted graph $G = (V,E)$ with a weight function $w : E \to R$ that associates a real value $w(e)$ to each edge $e \in E$ and a path $p = [v_0, v_1, ..., v_k]$, its weight $w(p)$ is the sum of the weights of edges that compose it, i.e. $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.
There are different variants regarding the shortest path problems as the case where there is only one destination and then the request is to find a path from each node $v \in V$ to the unique destination, at minimum cost. Other variants occur when the request is to look for a shortest path between a predetermined pair of nodes or between all pairs

of nodes. For more details about these problems we refer to the book [16].

2.3.1 All Color Shortest Path problem

The *All-Colors Shortest Path* (ACSP) is a combinatorial optimization problem, first introduced in [3]. The problem is defined on undirected graphs, in which a numerical attribute (weight) is associated to each edge, while a logical attribute (called *color*, or *label*) is given for each vertex. Therefore the different colors, appearing in the graph, partition the set of vertices into disjoint subsets. The aim of ACSP is to find the shortest possibly non-simple path, spanning each color of the graph; that is, each path composing a feasible solution needs to visit at least a vertex belonging to each color.

The problem can find application in different contexts. For instance, in a road network for the distribution of goods, vertices associated with the same color can represent different locations (warehouses or stores) in which specific types of goods can be picked up or stocked. Applications related to mobile sensor roaming and path planning are cited in [3].

Among similar problems presented in the literature, we recall the Shortest Path Tour (SPTP), the Forward Shortest Path Tour (FSPTP), the Generalized Traveling Salesman (GTSP) and the Generalized Minimum Spanning Tree (GMST). The SPTP ([29]) is a polynomially solvable optimization problem in which, given a source vertex s and a destination vertex d, the aim is to find a shortest path from s to d that crosses in a given sequence at least a vertex for each different color. Any node of the graph can be crossed while going from a color of the sequence to the following one. As for ACSP, the optimal solution can be non-simple. In the FSPTP variant ([6]), instead, the nodes associated to a given color can be visited only if at least a node of each preceding color has already been visited. Despite the similarities in the solution structure, the lack of predefined endpoints and of the predefined color visiting order make ACSP significantly harder to tackle than SPTP and FSPTP. In GTSP the aim is to find a minimum-cost hamiltonian tour that includes exactly a vertex for each different color. Therefore, in this case each vertex can be visited at most once, and the solution needs to be a cycle. GTSP is NP-Hard, and solution approaches have been mainly focused on Integer

Programming methods ([45], [30], [31]), heuristics ([63],[61],[4]) or transformations to reduce the problem to the classical TSP ([19]). Finally, in GMST, a tree spanning all different colors with minimum weight is sought. Two different variants, in which the tree is required to contain either exactly a vertex ([49],[27],[28],[34],[57],[52],[41]) or at least a vertex ([42],[22], [39]) for each different color have been proposed, both being NP-Hard and hard to approximate.

Chapter 3

Minimum spanning tree problem with conflicting edge pairs

3.1 Introduction

The Minimum Spanning Tree Problem with Conflicting Edge Pairs (MSTC) is a NP-Hard variant of the classical Minimum Spanning Tree problem. Given an undirected and edge weighted graph $G(V,E,P)$, where P is a set of conflicting edge pairs, MSTC problem consists of finding a minimum spanning tree of G without edges in conflict. The problem was introduced by Darmann et al. [17], [18]. In these works, the authors proved that MSTC problem is NP-Hard and that it is polynomially solvable when all the pairs in P are disjointed. Another polynomial case for MSTC problem occurs when the pairs in P satisfy the transitive property ([66]) that is: if $\{e_1,e_2\} \in P$ and $\{e_2,e_3\} \in P$ then even $\{e_1,e_3\}$ is in P.

In the literature, there are several optimization problems with conflict constraints such as the knapsack problem with conflict constraints [55], the maximum flow problems with disjunctive constraints [56], the bin packing problem with conflicts [58] and the minimum cost perfect matching with conflict pair constraints [53].

Regarding the MSTC resolution, in [66] the authors proposed several heuristic approaches and two exact algorithms based on Lagrangian relaxation. When a conflict free solution is not found, these heuristics return the number of conflict pairs present in the solutions. In [59] a branch and cut approach based on the conflict graph, i.e.

a graph where the set of nodes is E and the set of edges is P, was proposed. Moreover, the authors introduced a preprocessing phase that results very effective on some sets of instances. In this chapter we propose a multi ethnic genetic algorithm for the problem in which the fitness function is designed to simultaneously manage the two goals of the problem and three local search procedures to improve the solutions inside the population during the computation. Furthermore, we introduce a new set of valid inequalities for the problem, based on the properties of its feasible solutions, and we develop a Branch-and-cut algorithm based on them. The rest of the chapter is organized as follows. The problem is formally defined in section 3.2. The multi ethnic genetic algorithm we developed is described in section 3.3. The proposed Branch-and-Cut algorithm is described in section 3.4, while computational results are presented in section 3.5.

3.2 Notations and problem definition

Let $G(V,E,P)$ be an undirected edge weighted graph, where V is the set of vertices, E the set of edges and $P \subseteq E \times E$ is the set of *conflict edge pairs*. Formally:

$$P = \{\{e_i, e_j\} : e_i \in E, e_j \in E,\ e_i \text{ and } e_j \text{ are in conflict}\}$$

Since the couples in P are not ordered, $\{e_i, e_j\}$ and $\{e_j, e_i\}$ are the same couple. We denote by n and m the cardinality of V and E, respectively, and by w_{e_k} the non negative weight of the edge e_k.

Moreover, $\forall e_k \in E$ let $\mathbb{P}(e_k, E) = \{\{e_k, e_j\} \in P : e_j \in E\}$ be the set of conflict edge pairs containing the edge e_k and we indicate with $\chi(e_k)$ the set of edges that are in conflict with it. Furthermore, $\forall E' \subseteq E$ let $\zeta(E') = \bigcup_{e_k \in E'} \mathbb{P}(e_k, E')$ be the set of conflict edge pairs induced by edges in E'.

A *spanning tree* $T(V_T, E_T)$ of G is a connected subgraph of G such that $V_T = V$, $E_T \subseteq E$ and $|E_T| = n - 1$. The weight of T is denoted by $W(T)$ and it is given by the sum of edges weights in E_T while $\zeta(E_T)$ represents the set of conflict edge pairs present in T and $|\zeta(E_T)|$ the *number of conflicts* in T. When $|\zeta(E_T)| = 0$, we say that T is *conflict free*.

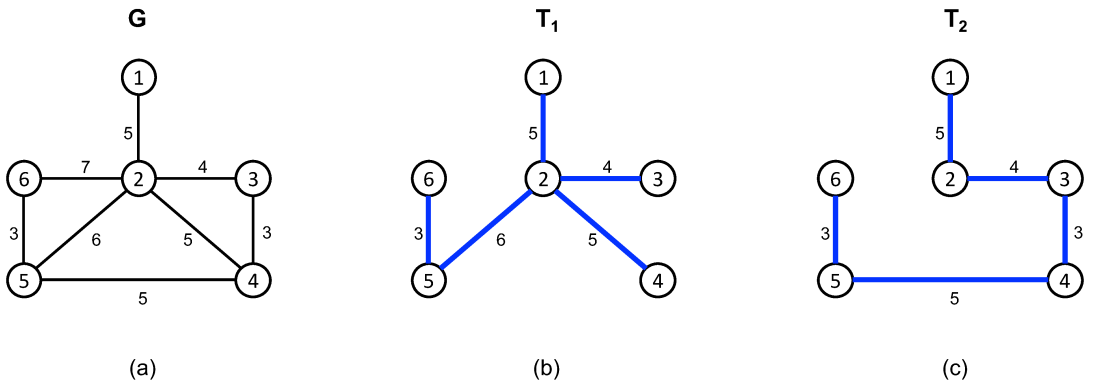


Figure 3.1: (a) A generic graph G. (b) A spanning tree T_1 of G with $W(T_1) = 23$. (c) A spanning tree T_2 of G with $W(T_2) = 20$

Given the graph G depicted in Figure 3.1(a), two spanning trees T_1 and T_2 of G are shown in Figure 3.1(b) and 3.1(c) with $W(T_1) = 23$ and $W(T_2) = 20$. A *minimum spanning tree* (MST) of G is any spanning tree of G with minimum weight.

The *minimum spanning tree problem with conflict constraints* (MSTC) consists of finding a minimum spanning tree T of G without conflicting edge pairs, that is $\forall\{e_i, e_j\} \in P$ at most one between e_i and e_j belongs to E_T. Obviously MSTC problem is infeasible on the graphs where there are not conflict free spanning trees. For instance, let us consider again graph G in Figure 3.1(a) and let $P = \{\{(1,2),(2,6)\},\{(1,2),(5,6)\}\}$. Since it is not possible to build a conflict free spanning tree of G, with the given set P, MSTC is infeasible on G.

3.3 A Multi-Ethnic Genetic Approach for the Minimum Conflict Weighted Spanning Tree Problem

It is worth noting that, by definition, the MSTC problem is infeasible on a graph G when there are not spanning trees without conflicts in G. Rather than marking an instance as infeasible, we prefer to solve a variant of MSTC problem in which the primary goal is to minimize the number of conflict edge pairs in the spanning tree T and, the secondary goal is to minimize the weight of T, if T has no conflicts. This new variant is named Minimum Conflict Weighted Spanning Tree problem (MCWST). For

instance, the trees T_1 and T_2, depicted in Figure 3.1, are two optimal solutions for the MCWST, with $|\zeta(E_{T_1})| = |\zeta(E_{T_2})| = 1$, when $P = \{\{(1,2),(2,6)\},\{(1,2),(5,6)\}\}$. Since these optimal solutions are not conflict free, their weight is neglected. On the contrary, if $P = \{\{(1,2),(2,6)\}\}$, both T_1 and T_2 are conflict free but T_2 is better than T_1 because $W(T_2) < W(T_1)$ (secondary goal).

From the definitions of MSTC and MCWST, it is easy to see that:

- T^* optimal solution of MSTC $\implies$ T^* optimal solution for MCWST;
- T^* optimal solution of MCWST and $|\zeta(E_{T^*})| = 0$ $\implies$ T^* optimal solution for the MSTC.

According to the previous observations, by addressing the MCWST problem, we solve even the MSTC problem while, for the instances on which MSTC problem is infeasible, we try to return a spanning tree with the minimum number of conflicts.

In this section, we propose a multi ethnic genetic algorithm for MCWST problem. In particular, we define a fitness function which is able to manage the two goals of the problem at the same time. Moreover, during the computation, we apply three local search procedures to improve the solutions within the population. Finally, we compare the multi-ethnic genetic algorithm with the heuristics proposed in [66] on their benchmark instances.

3.3.1 The Genetic Algorithm

In this subsection we introduce our genetic algorithm (GA) we have designed to solve the MCWST. In subsection 3.3.3, we describe how GA is embedded within a multi ethnic genetic framework to better explore the solution space and to improve its results.

Genetic algorithms, proposed for the first time by J. Holland in 1975 in his book Adaptation in Natural and Artificial Systems [40], are a family of metaheuristics based on the theory of Darwinian natural selection that regulates the biological evolution. While this theory works on a population of individuals, a genetic algorithm operates on a population of feasible solutions, called *chromosomes*, each of them composed by *genes*. The inefficiency of enumerating all feasible solutions recommends fixing

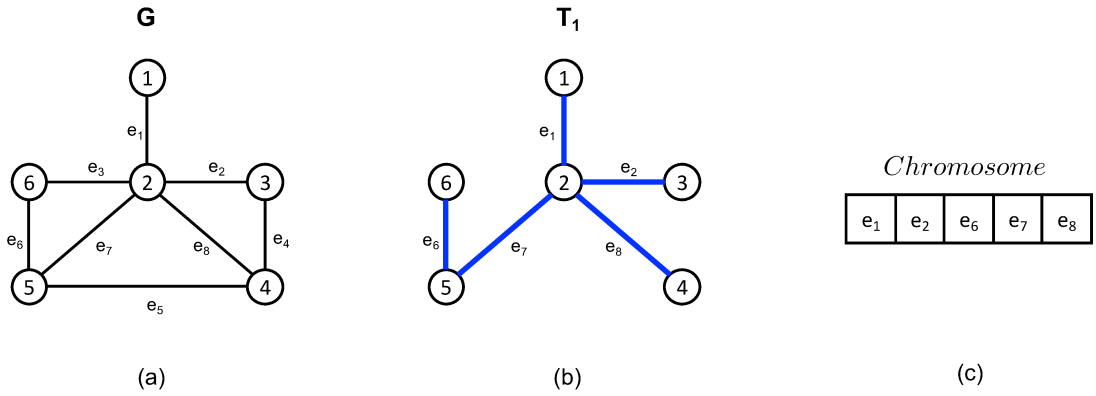


Figure 3.2: (a) A graph G. (b) A spanning tree T_1 of G. (c) The chromosome representing T_1.

in advance the size of solutions space, the *initial population*, on which the algorithm will act. The quality of each chromosome is evaluated by the *fitness function* that corresponds to the objective function or to a function strictly related to OF. The aim of the genetic algorithm is to find solutions as close as possible to the optimal one, by combining the chromosomes; this operation is, usually, carried out by crossover operator, which generate new chromosomes (children) by exchanging the genetic material (genes) of its parents. On the children is invoked the mutation operator that randomly changes one or more genes. This operator is fundamental to assure the diversity of the children from the parents.

The main elements of our genetic algorithm are described as follows.

Chromosome definition and fitness function The encoding of a feasible solution is the first step of any evolutive algorithm. In GA each chromosome T represents a spanning tree of G and it is encoded by using an array with $|V|-1$ positions (genes) each one containing an edge of E (see Figure 3.2). In the rest of chapter, we will use the terms spanning tree or chromosome interchangeably.

The quality of each chromosome T is evaluated by a fitness function f. Since the MCWST problem presents two different goals, we designed a fitness function that is able to manage both these goals simultaneously but respecting their priority. More in details, since the primary goal of MCWST is to minimize the number of conflicts, the lower is the number of conflicts into the chromosome, the better have to be its fitness

value. Moreover, given two or more conflict free chromosomes, the secondary goal states that the one with the lowest weight must have a better fitness value. The fitness function that satisfies the previous conditions is defined as follows:

$$f(T) = \begin{cases} |\zeta(E_T)| & \text{if } |\zeta(E_T)| > 0 \\ W(T) - W(T_{max}) & \text{otherwise} \end{cases} \tag{3.1}$$

where $W(T_{max})$ is an upper bound to the weight of any spanning tree of G. According to equation (3.1), lower the value of the fitness better the quality of the chromosome.

Note that the fitness value of any chromosome with conflicts is greater than zero while the fitness value of any conflict free chromosome is negative because $W(T_{max}) \geq W(T)$. As a consequence, any conflict free chromosome is always better than any chromosome with at least one conflict (primary goal). Moreover, the lower is $W(T)$, the lower will be its fitness value (secondary goal).

For instance, let us consider again the graph G in Figure 3.1(a) with P={{(1,2),(2,6)}}. An upper bound $W(T_{max})$ can be easily computed by adding the five highest edge weight of G obtaining $W(T_{max}) = 28$. According to the equation 3.1, $f(T_1) = W(T_1) - W(T_{max}) = 23 - 28 = -5$ while $f(T_2) = W(T_2) - W(T_{max}) = 20 - 28 = -8$ and therefore T_2 is better than T_1, as expected.

Initial population The initial population is composed by *SizePop* different chromosomes randomly generated. More in details, a random weight is assigned to each edge of G and then a minimum spanning tree of G is computed by using Prim's algorithm. If the chromosome obtained is already inside the population then it is rejected because no duplications are allowed. The procedure iterates until either SizePop different chromosomes are found or the threshold *maxD* is reached, where maxD denotes the maximum number of duplicate chromosomes that can be found before stopping the procedure. When this threshold is reached, SizePop is updated to the number of different chromosomes found so far.

The use of the MaxD threshold is necessary because it may happen that either there are no SizePop different spanning trees in G or it is very expensive to identify such trees through a random procedure.

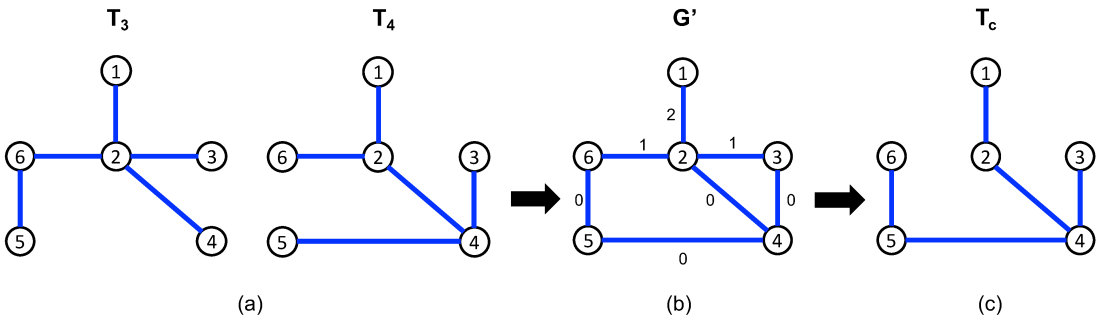


Figure 3.3: (a) Two spanning trees of G. (b) The subgraph G' generated by crossover. (c) The new child chromosome T_c.

Creation of new chromosomes *Selection procedure.* The selection procedure has the aim to ease the reproduction of individuals with better fitness and, at the same time, the aim to preserve the diversity of the population. In our implementation the selection of the parents is carried out by using a binary tournament strategy. This strategy consists of selecting randomly two chromosomes from the population. The one with the best fitness value is chosen as first parent T_{p1}. The same procedure is used to select the second parent T_{p2} assuring that T_{p2} is different from T_{p1}.

Crossover. The crossover operator is used to generate offspring by recombining the genes of selected parents in order to preserve the characteristics of their genetic heritage. The idea behind our crossover is to build a new spanning tree, by using the edges of parents, in which the number of conflicts is lower than the number of conflicts of parents. To this end, the crossover generates the subgraph $G'(V',E')$ of G induced by edges of both parents, i.e. $E' = E_{T_{p_1}} \cup E_{T_{p_2}}$. Then, the operator associates to each edge $e_i \in E'$ a weight equal to the number of conflicts in which e_i is involved with the other edges in E', i.e. $w(e_i) = |\mathbb{P}(e_i, E')|$. Finally, the crossover computes a minimum spanning tree T_c of G' as new child chromosome.

For instance, let us consider the graph G in Figure 3.2(a) and its two spanning tree, T_3 and T_4, depicted in Figure 3.3(a). Moreover, let us suppose that $P = \{\{(1,2),(2,3)\}, \{(1,2),(2,6)\}\}$. From T_3 and T_4 the crossover builds the subgraph G' induced by $E_{T_3} \cup E_{T_4}$ (Figure 3.3(b)) and it associates to each edge $e_i \in E'$ the weight $|\mathbb{P}(e_i, E')|$. Finally, the crossover computes a minimum spanning tree of G' obtaining the tree T_c

shown in Figure 3.3(c).

Mutation. In order to assure that the child chromosome T_c is different from parents, the mutation operator is applied on it. This operator randomly selects one edge in $E \setminus E_{T_c}$ and introduces this edge in T_c generating a cycle. To obtain a new tree, one of the edges in this cycle is randomly selected and removed. In this way, it is assured a differentiation between the child chromosome and the parents, reflecting the natural evolutionary process in which each genetic algorithm is inspired. This operation is carried out a number of times equal to 5% of $|V|$. However, if during the computation a conflict free chromosome is found, the mutation immediately stops and returns this chromosome.

Insertion and Stopping Criteria. If the child chromosome obtained after the mutation operator is already inside the population then it is rejected. Otherwise, the child chromosome will replace one of the $SizePop/2$ worst chromosomes in the population, selected in random way. As a consequence, the size of the population never changes and the best chromosome found, during the computation, never leaves the population. The genetic algorithm stops when a fixed number (*maxIt*) of iterations is reached.

3.3.2 Improvement procedures

In order to improve the best solution found by GA, we use three local search procedures named: *Conflicts Reduction Local Search* (CR), *Weight Reduction Local Search* (WR) and *Neighborhood Weight Reduction Local Search* (NWR). These procedures are invoked on all the chromosomes of the final population. Their aim is either to reduce the conflicts in the chromosomes or to reduce the weight of the conflict free chromosomes. In the following subsections the three procedures are described in details.

Conflicts Reduction Local Search The CR procedure is designed to reduce the number of conflicts in the chromosomes and then it is invoked only on the chromosomes with at least one conflict. Given a chromosome T, the first step of the procedure is to identify the edge $e_k \in E_T$ having the maximum number of conflicts with the other edges of E_T, i.e. $e_k = \underset{e_i \in E_T}{\operatorname{argmax}} |\mathbb{P}(e_i, E_T)|$.

The procedure removes e_k from E_T generating a forest composed by two subtrees T_1 and T_2. To obtain a new chromosome T', CR connects T_1 and T_2 by using the edge $e_r \in E \setminus E_T$ where $e_r = \underset{e_i \in E \setminus E_T}{\operatorname{argmin}} |\mathbb{P}(e_i, E_T \setminus \{e_k\})|$. If $|\zeta(E_{T'})| < |\zeta(E_T)|$ the procedure restart from T' otherwise it stops. The new chromosome obtained by CR, if any, replaces T in the population.

Weight Reduction Local Search The WR procedure is applied only on the conflict free chromosomes and it tries to minimize their weight without adding conflicts. Given a chromosome T, with $|\zeta(E_T)| = 0$, the procedure starts sorting, in ascending order, the edges in $E \setminus E_T$ according to their weights. Let $\mathbb{L}$ be the list of these sorted edges. At each iteration, the procedure selects from $\mathbb{L}$ the next edge e_k and if $|\mathbb{P}(e_k, E_T)| \leq 1$, it introduces e_k in E_T thus generating a cycle in T. In order to obtain a new chromosome it is necessary to break this cycle by removing one of its edges. There are two cases to consider here:

- $|\mathbb{P}(e_k, E_T)| = 0$
  Let e_j be the edge of the cycle with the maximum weight. Then e_j is removed from $E_T \cup \{e_k\}$ yielding a new conflict free chromosome T'. If $W(T') < W(T)$ then $T \leftarrow T'$, $\mathbb{L} \leftarrow \{e_j\} \cup \mathbb{L} \setminus \{e_k\}$ and WR restarts from the beginning of $\mathbb{L}$. Otherwise the procedure selects the next edge of $\mathbb{L}$.

- $|\mathbb{P}(e_k, E_T)| = 1$
  Let e_j be the edge in conflict with e_k in $E_T \cup \{e_k\}$. If e_j does not belong to the cycle then e_k is removed from the cycle, because no conflicts are allowed in this phase, and the procedure selects the next edge in $\mathbb{L}$. On the contrary, if e_j belongs to the cycle then it is removed yielding a new conflict free chromosome T'. If $W(T') < W(T)$ then $T \leftarrow T'$, $\mathbb{L} \leftarrow \{e_j\} \cup \mathbb{L} \setminus \{e_k\}$ and WR restarts from the beginning of $\mathbb{L}$. Otherwise the procedure selects the next edge of $\mathbb{L}$.

WR stops when all the edges in $\mathbb{L}$ have been selected and no improvements are obtained.

Neighborhood Weight Reduction Local Search The NWR is another procedure used to reduce the weight of conflict free chromosome. Given a conflict free chromosome T, the procedure generates a neighborhood of T as follows. For each edge

$e_k \in E \setminus E_T$ such that $|\mathbb{P}(e_k, E_T)| = 0$ or $|\mathbb{P}(e_k, E_T)| = 1$, NWR inserts e_k in E_T yielding a cycle. Now, if $|\mathbb{P}(e_k, E_T)| = 0$ then the procedure breaks the cycle by removing the edge e_j with maximum weight and it produces a new conflict free chromosome T_{e_k}. Otherwise, if $|\mathbb{P}(e_k, E_T)| = 1$, there is an edge $e_j \in E_T$ in conflict with e_k. If e_j belongs to the cycle then NWR removes e_j and it produces a new conflict free chromosome T_{e_k} otherwise no new chromosomes can be obtained in this iteration with the edge e_k. Then e_k is rejected and a new iteration is carried out with the next edge of $E \setminus E_T$. After the selection of all the edges in $E \setminus E_T$, the neighborhood of T is given by: $\mathcal{N}(T) = \bigcup_{e_k \in E \setminus E_T} T_{e_k}$. After the generation of $\mathcal{N}(T)$, NWR selects the chromosome $T' \in \mathcal{N}(T)$ with the minimum weight. If $W(T') < W(T)$ then $T \leftarrow T'$ and NWR generates the neighborhood of this new chromosome. Otherwise, the procedure stops.

Local search procedures framework The three local search procedures described above are applied on the chromosomes of the final population according to the rules shown in the diagram in Figure 3.4. More in details, let T^* be the best chromosome found by GA and let T be any chromosome of the final population. The following two cases are considered:

- $|\zeta(E_T)| = 0$
  In this case both the procedures, WR and NWR, are invoked on T yielding two new chromosomes T_1 and T_2, respectively. If the lowest fitness between $f(T_1)$ and $f(T_2)$ is better than $f(T^*)$ then T^* is updated accordingly.

- $|\zeta(E_T)| > 0$
  In this case the CR procedure is invoked on T yielding a new chromosome T'. If $|\zeta(E_{T'})| > 0$ but $f(T') < f(T^*)$ then we update T^* with T' and we proceed with the next chromosome into the population. Otherwise, if $|\zeta(E_{T'})| = 0$ the same steps of the previous case are carried out.

3.3.3 Multi Ethnic Genetic Approach

The multi ethnic genetic algorithm (Mega) is a technique developed for the genetic algorithms which is aimed at reducing the probability of remaining trapped at a local

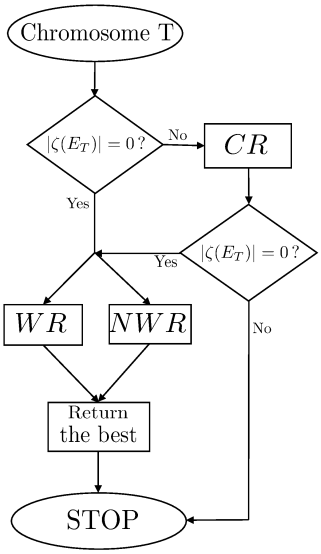


Figure 3.4: The improvement procedures framework applied on each chromosome of the final population.

minimum [12]. The main idea behind the algorithm is to split a starting population in k different sub-populations that, independently, evolve in k different environments. The resulting subpopulations are then recombined and the process is iterated, if necessary. Actually, the idea of allowing the simultaneously evolve of k different populations was already adopted in parallel genetic algorithms as the Island Model (see [46, 64]). However, in Mega each population is characterized by its own fitness function that is appropriately selected to diversify the evolution and to carry out a better exploration of the solution space (for more details see [12]).

In the following, we denote by $GA(\cdot)$ the application of our genetic algorithm with the fitness function.

Mega start from a single population P_t at time $t = 0$ and it evolves this population by using GA with the fitness function f (equation 3.1). After this evolutionary step, the obtained evolved population $\hat{P}_t$ is split into $k = 3$ different sub-populations $P_{t,1}$, $P_{t,2}$ and $P_{t,3}$. Differentiation of the environments is induced by slightly changing the fitness function for each sub-population. From f we retrieve three new fitness functions: f_1, f_2 and f_3. Each one of these functions randomly selects the 20% of edges in E_{T^*} and it applies a penalization on them. More in details, f_1 penalizes the selected edges of E_{T^*} by doubling their number of conflicts while f_2 penalizes them by doubling their

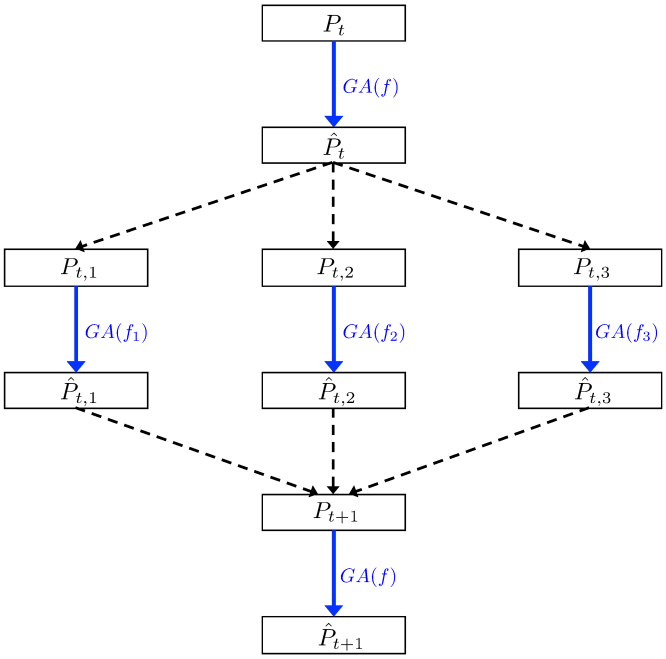


Figure 3.5: Multi ethnic genetic framework

weight. Finally, f_3 uses the two previous penalizations by doubling both the number of conflicts and the weight of selected edges.

Each of the sub-populations $P_{t,1}$, $P_{t,2}$ and $P_{t,3}$, in turn, evolves according to GA and to a fitness function f_i, with $i = 1, ..., 3$. The three resulting populations $\hat{P}_{t,1}$, $\hat{P}_{t,2}$ and $\hat{P}_{t,3}$ are merged into a unique population P_{t+1}. Finally, this population is evolved by using GA with f. The best chromosome T^* met during the computation is returned.

After a tuning phase, we set the parameters *SizePop* to 99 and *maxD* to 100. We use a variable value for *maxIt* according to the population on which GA is invoked. In particular, *maxIt* is equal to 2000 when GA is invoked on P_t and 1000 when it is invoked on P_{t+1}. Regarding the sub-populations, *maxIt* is equal to 100 because the size of this population is smaller than the size of P_t and because this value assure us an appropriate diversification of the evolutionary process.

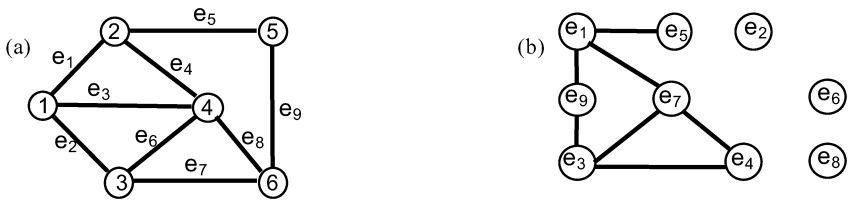


Figure 3.6: (a) An example of graph G with $|V| = 6$, $|E| = 9$ and conflicts set $P = \{\{e_1,e_5\},\{e_1,e_7\},\{e_1,e_9\},\{e_3,e_4\},\{e_3,e_7\},\{e_3,e_9\},\{e_4,e_7\}\}$ ($|P| = 7$). (b) The related conflict graph $G' = (E,P)$, where each node corresponds to an edge of G and each edge corresponds to a pair in P.

3.4 A Branch-and-Cut approach for the MSTC problem

In this section, we propose a Branch-and-Cut approach to solve the Minimum Spanning Tree problem with conflicting edge pairs. In particular we introduce a new set of valid inequalities for the problem, based on the properties of its feasible solutions, and we develop a Branch-and-Cut algorithm based on them. The first authors to face the problem through an exact approach have been Samer and Urrutia in [59]. They presented a mathematical model for the MSTC problem, as well as two sets of valid inequalities. In order to introduce these new sets of valid inequalities, the authors gave an equivalent definition of the problem by defining the concept of *conflict graph* $G' = (E,P)$. G' contains a node for each edge E of G, and two nodes e_i, e_j are connected in G' if and only if $\{e_i,e_j\} \in P$.

Figure 3.6 shows an example of graph G and the related conflict graph G'. We can note that, for instance, $\{e_1,e_3,e_5,e_6,e_8\}$ is a feasible MST solution being a spanning tree of G, but it is not feasible for the MSTC since $\{e_1,e_5\} \in P$. On the other hand, $\{e_2,e_4,e_5,e_6,e_9\}$ is a conflict free spanning tree and therefore it is a feasible MSTC solution.

In this section we test the effectiveness and performance of our approach on a set of instances originally proposed in [66]. We compared these results with those obtained by the exact algorithm presented in [59], that was tested on the same dataset. Furthermore, we also test our approach on a new, wider set of instances that we generated.

3.4.1 Basic Mathematical Model

In this subsection we present a mathematical model for the MSTC problem, based on a traditional Subtour Elimination formulation for the MST with the additional constraints to avoid the conflicts. This model was also considered in [59]. The formulation only uses a type of decision variables x_e associated with the edges of G, with the following meaning:

$$x_e = \begin{cases} 1 & \text{if } e \text{ is selected} \\ 0 & \text{otherwise} \end{cases}$$

The mathematical programming formulation of the MSTC is the following one:

$$(\textbf{ILP}) \quad \min \sum_{e \in E} w_e x_e \tag{3.2}$$

s.t.

$$\sum_{e \in E} x_e = |V| - 1, \tag{3.3}$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \qquad \forall S \subseteq V, S \neq \emptyset, \tag{3.4}$$

$$x_{e_i} + x_{e_j} \leq 1, \qquad \forall \{e_i, e_j\} \in P, \tag{3.5}$$

$$x_e \in \{0,1\}, \qquad \forall e \in E. \tag{3.6}$$

The objective function (3.2) minimizes the weight of the spanning tree. Constraint (3.3) imposes the selection of $n-1$ edges (recall that $|V| = n$) while Constraints (3.4) are the classical subtour elimination constraints. Finally, Constraints (3.5) ensure that two edges in conflict cannot be simultaneously selected in the solution while constraints (3.6) are the integrality constraints.

3.4.2 Valid inequalities

In this subsection we present three classes of valid inequalities for the MSTC that we used to design a Branch-and-Cut approach for this problem. The first class, named

degree-cut inequalities, assure that there are not isolated vertices in the solution; we use them to enforce the Subtour Elimination model. The second one, the *conflict-cycle* inequalities combine the request of avoiding both cycles and conflicts and represent our main contribution. Finally, the third class of inequalities are the well known *odd-cycle* inequalities that are derived from the conflict graph structure. In the following subsection we describe in details these valid inequalities.

The degree-cut inequalities Since the solution of the MSTC is a spanning tree then for each node we have at least one incident edge selected. For this reason, we add to our model the following valid inequalities:

$$\sum_{e \in \delta(v)} x_e \geq 1, \qquad \forall v \in V. \tag{3.7}$$

The constraints (3.7) state explicitly that the degree of any node into the solution must be greater than or equal to 1. These inequalities improve the relaxed solution value of ILP model. Indeed, by removing the constraints (3.4) from ILP model, the optimal solution is obtained by selecting the cheapest $n-1$ edges of the graph. This could lead to the presence of isolated nodes (i.e. with degree equal to zero) in the solution. The inequalities (3.7) prevent the construction of these type of solutions.

Since the number of inequalities (3.7) is equal to n, no separation procedures are applied but they are directly introduced into the ILP model as a priori constraints. Obviously, these constraints are not necessary to represent the solutions space but, in our experiments, they speed up the convergence of our Branch-and-Cut.

Conflict-cycle inequalities The conflict-cycle inequalities are a stronger version of the subtour elimination constraints obtained by exploiting the conflicts among the edges.

Let Ψ be a set of edges that generate a cycle in G, and let us suppose that two of these edges are in conflict with another edge e_c that does not belong to Ψ. Then, in any feasible solutions of MSTC, the number of edges of $\Psi \cup \{e_c\}$ must be lower than or equal to $|\Psi| - 1$. The following theorem proves that these inequalities are valid for the MSTC.

Theorem 3.4.1 *Let Ψ be a cycle of G and let e_c be an edge outside this cycle that is in conflict with two edges of Ψ. Then the constraint*

$$\sum_{e_i \in \Psi} x_{e_i} + x_{e_c} \leq |\Psi| - 1, \tag{3.8}$$

is a valid inequality for the MSTC problem.

proof 3.4.1 *By contradiction, let us suppose that in a feasible solution of MSTC we have:*

$$\sum_{e_{i'} \in \Psi'} x_{e_{i'}} + x_{e_g} > |\Psi'| - 1,$$

where $\Psi' \subseteq E$ is a cycle of G, $e_{j'}, e_{k'} \in \Psi'$, $e_g \in E \setminus \Psi'$, and $e_{j'}, e_{k'} \in \chi(e_g)$. We have to consider the following two cases:

- *If $x_{e_g} = 0$ then $\sum_{e_{i'} \in \Psi'} x_{e_{i'}} > |\Psi'| - 1$. However, this last condition violates Constraints (3.4). A contradiction.*
- *if $x_{e_g} = 1$ then*

$$\sum_{e_{i'} \in \Psi'} x_{e_{i'}} + 1 > |\Psi'| - 1 \quad \Rightarrow \quad \sum_{e_{i'} \in \Psi'} x_{e_{i'}} > |\Psi'| - 2.$$

  Due to this last condition at least one of variables $x_{e_{j'}}$ and $x_{e_{k'}}$ must be equal to 1, thereby violating the Constraints (3.5).

Inequalities of type (7) are called conflict-cycle inequalities.

In Figure 3.7 an example of how the inequalities (3.8) work is shown. Figure 3.7(a) is the initial graph. Notice that the solution in Figure 3.7(b) satisfies the classical subtour elimination constraints, while it is cut off by inequalities (3.8). Indeed, considering the cycle $\Psi = \{e_4, e_5, e_9, e_8\}$ (Figure 3.7(c)), we note that e_5 and e_9 belong to $\chi(e_1)$ (see Fig. 3.6).

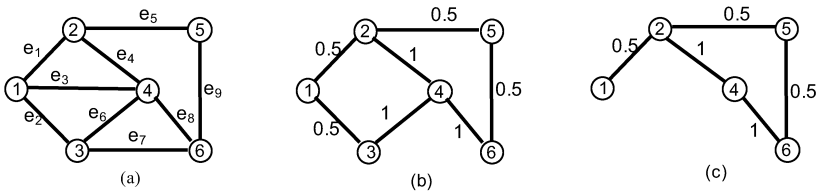


Figure 3.7: (a) The input graph G. (b) A feasible solution that satisfies constraints (2)-(4) and the inequalities $0 \leq x_e \leq 1, \forall e \in E$. (c) Considering the cycle $\Psi = \{e_4, e_5, e_9, e_8\}$ in the solution (b) and edge e_1 in conflict with e_5 and e_9, the related constraint (7) is violated.

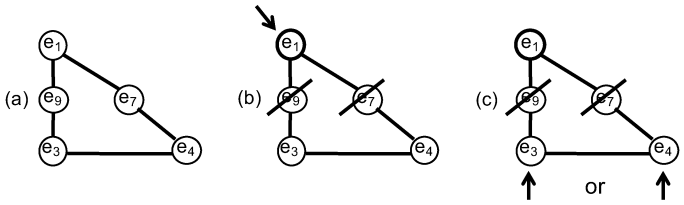


Figure 3.8: (a) An odd-cycle of length 5 in the conflict graph G' in Figure 3.6. (b) If we choose e_1, it is not possible to choose e_5 and e_9. (c) At this point, only one between e_3 and e_4 can be part of a MSTC solution.

Odd-Cycle inequalities Another set of valid inequalities for the MSTC are the well-known odd-cycle inequalities. These inequalities are based on the conflict graph G' described previously. Each vertex of G' is associated to an edge of G and two nodes are connected if the corresponding edges of G are in conflict. This means that the selection of two connected vertices in G' is equivalent to select two edges in conflict in G. For this reason, given a cycle C' of G', having an odd number k of edges, it is easy to see that it is possible to select at most $\frac{k-1}{2}$ vertices of the cycle (that is, edges of G) without violating the conflict constraints. Formally,

$$\sum_{e \in C'} x_e \leq \frac{|C'|-1}{2}, \qquad \forall C' \subseteq E \quad odd-cycle \quad in \quad G' \tag{3.9}$$

In Figure 3.8 we show that, given an odd cycle of length 5 in the conflict graph of the example in Figure 3.6, the maximum number of edges that can be chosen is

$\frac{5-1}{2} = 2$.

A Branch-and-Cut approach based on the ILP model and using, among the others, the odd-cycle inequalities was presented in [59]. In the computational test section we will carry out a comparison between our Branch-and-Cut approach and theirs.

3.4.3 Branch-and-Cut approach

In this subsection, we outline the main ingredients of our Branch-and-Cut algorithm for the optimal MSTC solution as well as the separation procedures for the valid inequalities described in previous subsection. To obtain upper bounds that help pruning the search tree, we use the algorithm described in section 3.3. However, since it is known that even finding a feasible MSTC solution is NP-hard, there are several instances where these upper bounds are not available because the multi-ethnic genetic algorithm did not found them within a fixed time limit.

Initial relaxation The initial relaxation of ILP, named $\mathscr{R}(ILP)$, is composed by constraints (2),(4),(6) and the inequalities $0 \leq x_e \leq 1$.

Separation procedures The odd-cycle inequalities are separated by using the exact algorithm proposed in [33] while the subtour elimination constraints are separated by using the exact algorithm presented in [54].

In the following, we describe our procedure to separate conflict-cycle inequalities (3.8). Given a solution $\bar{x}$ of $\mathscr{R}(ILP)$, we build a new graph $\tilde{G} = (V, \tilde{E})$ where $\tilde{E} = \{e = (i,j) \in E : \bar{x}_e > 0\}$. To each edge $\tilde{e} \in \tilde{E}$ the weight $w_{\tilde{e}} = 1 - \bar{x}_{\tilde{e}}$ is assigned. The conflict-cycle inequalities (3.8) are heuristically separated by using the graph $\tilde{G}$ with the following procedure. Given any couple of nodes $\tilde{v}_1, \tilde{v}_2 \in V$ such that $(\tilde{v}_1, \tilde{v}_2) \in \tilde{E}$, we look for the shortest path between them in $\tilde{G}$ which does not include the edge $(\tilde{v}_1, \tilde{v}_2)$. If such a path exists, we append $(\tilde{v}_1, \tilde{v}_2)$ to it, obtaining a cycle $\tilde{\Psi} \subseteq \tilde{E}$. To individuate a violated inequality, we look for an edge $\tilde{e}_3 \in \chi(\tilde{e}_1) \cap \chi(\tilde{e}_2) \setminus \tilde{\Psi}$ where $\tilde{e}_1, \tilde{e}_2 \in \tilde{\Psi}$ and such that $\sum_{\tilde{e} \in \tilde{\Psi}} \bar{x}_{\tilde{e}} + \bar{x}_{\tilde{e}_3} > |\tilde{\Psi}| - 1$; hence we look for all possible edges of this type and all the violated inequalities are introduced in the model. Note that if $\sum_{\tilde{e} \in \tilde{\Psi}} \bar{x}_{\tilde{e}} + 1 \leq |\tilde{\Psi}| - 1$, it is impossible to find violated inequalities of the type (3.8), hence we don't look for them in this case. Furthermore, we decided to use an

additional tolerance parameter $\varepsilon_c \geq 0$, meaning that we only consider violated inequalities if $\sum_{\tilde{e}\in\tilde{\Psi}} \bar{x}_{\tilde{e}} + \bar{x}_{\tilde{e}_3} > |\tilde{\Psi}| - 1 + \varepsilon_c$. The computational complexity of this algorithm is $O(m^2 logn)$. In fact the individuation of a shortest path requires $|\tilde{E}|log|V|$ and it is invoked for each edge in $\tilde{E}$. We use an m x m binary matrix to state in $O(1)$ that two edges are in conflict.
Note that the separation procedure for the subtour elimination constraints cannot be used for inequalities (3.8), because it is not sufficient to individuate the set of vertices S that generate a cycle. We need to know what are the edges of the cycle to separate the conflict-cycle inequalities.

Cutting plane phase At each iteration of the cutting-plane algorithm:

- if the variables in the LP solution are all integer, the subtour elimination constraints (3.4) are heuristically separated through a DFS procedure;
- otherwise, the following separation procedures are used:
  1. Exact separation procedure [54] for the subtour elimination constraints (3.4).
  2. Heuristic algorithm for separating the conflict-cycle inequalities (3.8) with $\varepsilon_c = 0.1$.
  3. Exact algorithm for separating the odd-cycle inequalities (3.9) only at the root node.

If all separation procedures fail to find violated inequalities or a tailing-off criterium is met, we branch on variables using the default parameters of CPLEX. The tailing-off is applied when the improvement in the upper bound is less than 10^{-5} in five consecutive iterations.
To keep the size of the LP as small as possible, in each node of the search tree we never add more than 50 valid inequalities. The value of this parameter was chosen after a preliminary tuning phase.

3.5 Computational results

In this subsection we present the computational results of the tests we made in order to evaluate the performance and effectiveness of our two proposed approaches.

3.5.1 Mega

The algorithm was coded in C++ on an OSX platform (Imac mid 2011), running on an Intel Core i7-2600 3.4 GHz processor with 8 GB of RAM.

Mega was tested on the benchmark instances presented in [66] and was compared with the tabu search (TS) algorithm proposed in the same paper. These benchmark instances were generated by using a different value for nodes, edges and number of conflicts and they are classified into two types: *type 1* and *type 2*. By construction, in all type 2 instances is present at least one conflict free solution while for the type 1 instances this is not assured. In order to have a fair comparative study, the CPU time reported in [66] have been scaled according to the Whetstone benchmarks [1].

We now present the results of our tests. The first experiment we carried out is aimed at verifying the stability of Mega by running the algorithm five times on each instance and by comparing the best and average values found. Results are shown in Table 3.1 that is organized as follows.

The first block of rows is related to instances of Type 1 (instances 1-23) while the second block is related to instances of type 2 (instances 24-50). The first four columns report data on the instances: the identifier (*id*), the number of nodes (n), the number of edges (m) and the cardinality of P (p). The following two columns, *AvgWeight* and *AvgConf*, show the average weight and the average number of conflicts computed on the five runs while the next two columns report the weight (*Weight*) and the number of conflict (*Conf*) of the best solution found on the five runs. We remind that the best solution is the one with the minimum number of conflicts and, if this solution has zero conflicts, with the minimum weight of the tree. Note that, for the solutions with conflicts, we do not report the weight because its value loses meaning in these cases. Finally, the last column *Gap* shows the percentage gap between AvgConf and Conf. This gap is computed with the following formula: $100 \times \frac{Conf - AvgConf}{AvgConf}$.

Let us start the comparison on the instance of type 1. The primary goal is to minimize the number of conflicts. From this point of view, AvgConf and Conf are both equal to zero on the instances 1-9, except 6, and instance 16. Conf is equal to 0 even on the instances 6 and 13 while AvgConf is equal to 0.6 and 2.2 on them. This is an acceptable difference. Unfortunately, the gap value on these last two instances loses

	id	n	m	p	AvgWeight	AvgConf	Weight	Conf	Gap
Type 1	1	50	200	199	708	0.0	708	0.0	0.00%
	2	50	200	398	770	0.0	770	0.0	0.00%
	3	50	200	597	917	0.0	917	0.0	0.00%
	4	50	200	995	1365.4	0.0	1336	0.0	0.00%
	5	100	300	448	4099.2	0.0	4088	0.0	0.00%
	6	100	300	897	-	0.6	6095	0.0	-100.00%
	7	100	500	1247	4291.2	0.0	4275	0.0	0.00%
	8	100	500	2495	6325	0.0	6199	0.0	0.00%
	9	100	500	3741	7788	0.0	7665	0.0	0.00%
	10	100	300	1344	-	10.4	-	10.0	-3.85%
	11	100	500	6237	-	9.4	-	8.0	-14.89%
	12	100	500	12474	-	37.8	-	35.0	-7.41%
	13	200	600	1797	-	2.2	15029	0.0	-100.00%
	14	200	600	3594	-	60.0	-	57.0	-5.00%
	15	200	600	5391	-	143.2	-	142.0	-0.84%
	16	200	800	3196	22350.8	0.0	22110	0.0	0.00%
	17	200	800	6392	-	27.6	-	23.0	-16.67%
	18	200	800	9588	-	88.6	-	87.0	-1.81%
	19	200	800	15980	-	177.2	-	172.0	-2.93%
	20	300	800	3196	-	55.0	-	52.0	-5.45%
	21	300	1000	4995	-	23.4	-	21.0	-10.26%
	22	300	1000	9990	-	180.6	-	176.0	-2.55%
	23	300	1000	14985	-	330.2	-	329.0	-0.36%
Type 2	24	50	200	3903	1636	0.0	1636	0.0	0.00%
	25	50	200	4877	2043	0.0	2043	0.0	0.00%
	26	50	200	5864	2338	0.0	2338	0.0	0.00%
	27	100	300	8609	7434	0.0	7434	0.0	0.00%
	28	100	300	10686	7968	0.0	7968	0.0	0.00%
	29	100	300	12761	8166	0.0	8166	0.0	0.00%
	30	100	500	24740	12652	0.0	12652	0.0	0.00%
	31	100	500	30886	11232	0.0	11232	0.0	0.00%
	32	100	500	36827	11481	0.0	11481	0.0	0.00%
	33	200	400	13660	17728	0.0	17728	0.0	0.00%
	34	200	400	17089	18617	0.0	18617	0.0	0.00%
	35	200	400	20470	19140	0.0	19140	0.0	0.00%
	36	200	600	34504	20716	0.0	20716	0.0	0.00%
	37	200	600	42860	18025	0.0	18025	0.0	0.00%
	38	200	600	50984	20864	0.0	20864	0.0	0.00%
	39	200	800	62625	39895	0.0	39895	0.0	0.00%
	40	200	800	78387	37671	0.0	37671	0.0	0.00%
	41	200	800	93978	38798	0.0	38798	0.0	0.00%
	42	300	600	31000	43721	0.0	43721	0.0	0.00%
	43	300	600	38216	44267	0.0	44267	0.0	0.00%
	44	300	600	45310	43071	0.0	43071	0.0	0.00%
	45	300	800	59600	43125	0.0	43125	0.0	0.00%
	46	300	800	74500	42292	0.0	42292	0.0	0.00%
	47	300	800	89300	44114	0.0	44114	0.0	0.00%
	48	300	1000	96590	71562	0.0	71562	0.0	0.00%
	49	300	1000	120500	76345	0.0	76345	0.0	0.00%
	50	300	1000	144090	78880	0.0	78880	0.0	0.00%

Table 3.1: Best and average values found by Mega on the type 1 and type 2 instances.

meaning because, for any value greater than zero of AvgConf, this gap is always equal to 100%. On the remaining 12 instances, Gap is greater than 5% only 5 times. Regard-

ID	n	m	p	Opt/Best	TS		Mega		GapOpt	GapTS
					Weight	Time	Weight	Time		
1	50	200	199	708	711	1.17	708	0.71	0.00%	-0.42%
2	50	200	398	770	785	1.13	770	0.68	0.00%	-1.91%
3	50	200	597	917	1086	0.98	917	0.63	0.00%	-15.56%
4	50	200	995	1324	1629	1.16	1336	0.66	0.91%	-17.99%
5	100	300	448	4041	4207	6.33	4088	2.39	1.16%	-2.83%
6	100	300	897	6523*	-	5.99	6095	1.81	-6.56%	-
7	100	500	1247	4275	4539	17.71	4275	5.18	0.00%	-5.82%
8	100	500	2495	6653*	6812	17.09	6199	5.12	-6.82%	-9.00%
9	100	500	3741	-	8787	14.94	7665	3.72	-	-12.77%

Table 3.2: Type 1 Problems-Feasible.

ing the weight values, AvgWeight and Weight have the same value on the first three instances. The maximum gap value is equal to 2.15% and it occurs on the instance 4. In all other cases, the percentage gap is lower than 2%.

On all the instances of type 2, the best and average solutions coincide and these solutions are all conflict free. These results show that type 2 instances are much easier to solve than type 1 instances. A conclusion already reported in [66].

Summarizing, from the results of Table 3.1 we retrieve that on 30 out of 50 instances the best and the average solutions of Mega coincide in terms of both number of conflicts and weight value. On the remaining 20 instances, the gap on the number of conflicts is only five times greater than 5%, ruling our the two "special cases" of instances 6 and 13 where the value is 100% because Conf is equal to zero. Finally, on the few instances where the weight value is different, this gap is very low. According to these results, we can conclude that Mega has a good stability level.

In the next three tables, we compare the results of Mega with the tabu search TS proposed in [66]. Following [66], Table 3.2 contains the subset of type 1 instances on which the TS found a conflict free solution. Unfortunately, for the instance 6, only the CPU time but not the weight is reported.

The first four columns show the characteristics of the instance as already mentioned for Table 3.1. The column Opt/Best reports the optimal solution value found by mathematical model described in [66] and implemented in CPLEX. A time limit of 5000 seconds is fixed and whenever CPLEX reaches this limit, the solution value

is reported with the symbol "*". This means that CPLEX did not certify the optimality of the solution and the value reported is an upper bound of the optimal solution. Moreover, if CPLEX does not find a feasible solution, within the time limit, the symbol "-" is shown. The next four columns report the weight (*Weight*) of the tree and the CPU time (*Time*), in seconds, of TS and Mega, respectively. Since both the algorithms always find conflict free solutions on this set of instances, we do not report a column with the number of conflicts. The last two columns show the percentage gap between the weight of Mega and the optimal solution (*GapOpt*) and between the weight of Mega and TS (*GapTS*). These percentage gaps are computed with the formulas: $100 \times \frac{Mega(Weight)-Opt}{Opt}$ and $100 \times \frac{Mega(Weight)-TS(Weight)}{TS(Weight)}$, respectively.

By comparing the solutions of Mega with the optimal ones, we can see that our algorithm optimally solves the instances 1, 2, 3 and 7 while GapOpt is equal to 0.91% and 1.16%, on the instances 4 and 5, respectively. On the instances 6 and 8, the solution values of Mega are significantly lower than the upper bounds found by CPLEX. Finally, on the instance 9, Mega finds a feasible solution in 4 seconds while CPLEX did not in 5000 seconds. These results prove the effectiveness of Mega because, often, it finds the optimal solution or a solution very close to the optimal one.

The results of Table 3.2 show that Mega overcomes TS in terms of computational time and solution quality. Indeed, the solutions found by Mega are always better than the solutions found by TS with a percentage gap that ranges from 0.42% (instance 1) to 17.99% (instance 4). In particular, on 5 out of 8 instances GapTS is greater than 5.8%. Finally, TS never finds an optimal solution while Mega does it four times. The computational time spent on these instances is low for both algorithms but Mega is always faster than TS.

Table 3.3 shows the results of GA and TS on the remaining type 1 instances on which TS never finds conflict free solutions. The first four columns show the characteristics of the instance. The next four columns report the number of conflicts (*Conf*) and the CPU time (*Time*) of TS and Mega, respectively. The column *Gap* shows the percentage gap between the Conf values of two algorithms. This percentage gap is computed with the formula $100 \times \frac{Mega(Conf)-TS(Conf)}{TS(Conf)}$.

We remind that for the type 1 instances it is not guaranteed the presence of a conflict free spanning tree. However, Mega certified the presence of a conflict free solution on two of these instances (13 and 16) while TS finds solutions with two conflicts on these

id	n	m	p	TS		Mega		Gap
				Conf	Time	Conf	Time	
10	100	300	1344	13	6.77	10	2.69	-23.08%
11	100	500	6237	11	15.17	8	4.98	-27.27%
12	100	500	12474	41	14.64	35	6.68	-14.63%
13	200	600	1797	2	72.48	0	12.23	-100.00%
14	200	600	3594	67	70.24	57	21.71	-14.93%
15	200	600	5391	149	80.12	142	29.43	-4.70%
16	200	800	3196	2	105.21	0	23.42	-100.00%
17	200	800	6392	39	98.01	23	28.20	-41.03%
18	200	800	9588	95	97.10	87	35.32	-8.42%
19	200	800	15980	178	104.93	172	44.48	-3.37%
20	300	800	3196	63	239.63	52	62.68	-17.46%
21	300	1000	4995	38	303.04	21	83.68	-44.74%
22	300	1000	9990	207	345.25	176	117.58	-14.98%
23	300	1000	14985	351	381.28	329	134.42	-6.27%
Avg				**89.7**	**138.1**	**79.4**	**43.4**	

Table 3.3: Type 1 Problems-Feasible Unknown.

instances. Behind these two cases, it is evident that Mega is more effective than TS because it always finds better solutions. Ruling out the instances 13 and 16, on the remaining 12 instances the Gap value ranges from 3.37% to the 44.74% and this gap is 8 times greater than 14%.

Regarding the performance, Mega is always faster than TS. More in details, Mega solves all the instances up to 200 nodes in less than a minute while, in the same time, TS solves only the first three instances. The maximum time spent by Mega is 135 seconds on the instance 23 while TS requires 381 seconds on the same instance. According to the average values reported on the last line of the table, we can state that Mega is three times faster than TS and the solutions provided by Mega are 11.5% better than the solutions of TS.

The last comparison is carried out on the type 2 instances and the results are shown in Table 3.4. On these instances both the algorithms always find the optimal solution. All these optimal solutions are conflict free and their weight is reported into the column Opt. For this reason, we compare only the CPU times in this table. The first four columns are the same of the previous table. The next three columns report the optimal

id	n	m	p	Opt	TS	Mega	Gap
24	50	200	3903	1636	1.32	0.46	-65.15%
25	50	200	4877	2043	1.93	0.47	-75.69%
26	50	200	5864	2338	1.56	0.51	-67.36%
27	100	300	8609	7434	8.03	1.88	-76.58%
28	100	300	10686	7968	7.47	1.68	-77.51%
29	100	300	12761	8166	7.88	1.72	-78.18%
30	100	500	24740	12652	19.29	3.30	-82.90%
31	100	500	30886	11232	16.77	3.48	-79.24%
32	100	500	36827	11481	15.16	3.51	-76.85%
33	200	400	13660	17728	30.27	7.25	-76.05%
34	200	400	17089	18617	38.63	7.49	-80.61%
35	200	400	20470	19140	26.64	7.40	-72.22%
36	200	600	34504	20716	82.49	11.17	-86.46%
37	200	600	42860	18025	96.16	11.35	-88.20%
38	200	600	50984	20864	122.67	12.38	-89.91%
39	200	800	62625	39895	117.33	16.35	-86.07%
40	200	800	78387	37671	106.52	15.70	-85.26%
41	200	800	93978	38798	105.42	16.02	-84.80%
42	300	600	31000	43721	112.03	18.61	-83.39%
43	300	600	38216	44267	153.88	21.28	-86.17%
44	300	600	45310	43071	98.99	24.32	-75.43%
45	300	800	59600	43125	214.50	31.86	-85.15%
46	300	800	74500	42292	191.63	34.31	-82.10%
47	300	800	89300	44114	245.12	34.25	-86.03%
48	300	1000	96590	71562	301.27	39.81	-86.79%
49	300	1000	120500	76345	287.49	31.60	-89.01%
50	300	1000	144090	78880	325.16	36.11	-88.89%
Avg					**101.32**	**14.60**	

Table 3.4: Type 2 Problems.

weight (Opt) and the CPU time of TS and Mega, respectively. Finally, the column *Gap* shows the percentage gap values of the CPU times.

It is evident from the values of column Mega that type 2 instances are easier to solve than type 1 instances because all these instances are optimally solved in less than 37 seconds. Once again, Mega results always faster than TS with a percentage gap that is always greater than 65%. In particular, this gap grows as the size of instance

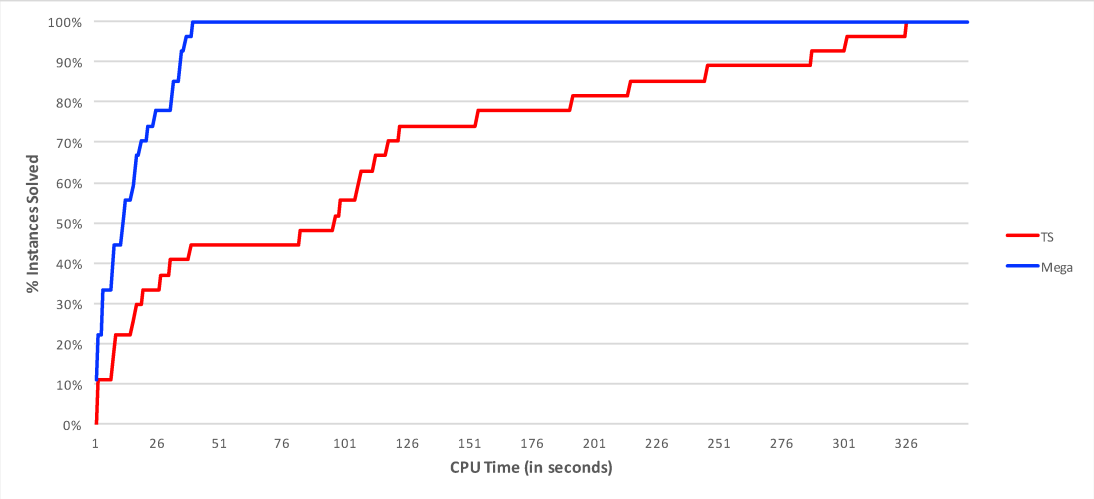


Figure 3.9: Performance comparison of TS vs Mega from the results of Table 3.4.

grows. Indeed, on the instances with 200 or more nodes Gap is almost always greater than 80%. The average values reported on the last line of the table show the relevant difference between the two algorithms from the performance point of view.

Finally, in Figure 3.9 we represent the results of Table 3.4 in a way that better highlights the performance of two algorithms. The horizontal axis represents the CPU time in seconds and the vertical axis represents the percentage of instances solved within a fixed CPU time. This means that as faster is the curve growth as better is the performance of the algorithm. The blue curve is associated to Mega and the red curve to TS. It is evident from this figure that the blue curve grows much faster than the red curve. Indeed, Mega reaches the 100% of instances solved in around 36 seconds. In the same time (36 seconds) TS solves around 45% of the instances while it requires 326 seconds to reach the 100% of solved instances.

3.5.2 Branch-and-Cut approach

In this subsection we present the computational results of the tests we made in order to evaluate the performance of our Branch-and-Cut algorithm (from now on called BC). The algorithm was coded in C++ on an OSX platform (iMac, mid 2011), running on an Intel(R) Core(TM) i7-2600 CPU 3.40GHz (family 6, model 42, stepping 7) with 8 GB of RAM, equipped with the IBM ILOG CPLEX 12.6.1 solver (single thread mode).

We compared the results of BC with the Branch-and-Cut algorithm (from now on called SU) proposed in [59]. Following [59], for all the experiments we considered a time limit equal to 5000 seconds. Furthermore, we considered a memory limit set to 3 GB. In this previous work, the authors propose a preprocessing procedure to simplify the instances before solving them. They divided the instances in two subsets, namely type 1 and type 2. Instances belonging to type 2 resulted to be very easy to solve after the preprocessing phase. Indeed, the authors do not present results about the SU performances on these instances, since they state that after this preprocessing (taking up to 18 seconds) all instances of this group were solved in negligible time. On these same instances, the genetic algorithm that we used to initialize our method always found (in up to 26 seconds) solutions that were very quickly certified to be optimal by BC. For these reasons, we compare our results only on the harder type 1 instances. We want to remark that we did not apply any preprocessing before solving them with BC. As will be shown, despite this, we obtained better results in all cases except one. This result, in our opinion, emphasizes the effectiveness of our algorithm.

Table 3.5 reports the results of the comparison between BC and SU.

The first four columns of the table show the information concerning the instance: a numerical identifier (*id*), the number of nodes (n), of edges (m) and of conflict pairs (p). The next two columns report the lower (LB) and upper (UB) bounds found by SU. When the lower and the upper bounds coincide, i.e. an optimal solution is found, the optimal value is reported between the LB and UB columns. When a "-" is reported, no feasible solution has been found. Finally, the last three columns report the lower bound, the upper bound and the computational time (*Time*), in seconds, of BC. The bounds are shown in bold whenever the solution found by BC is better than the solution found by SU. In [59] the authors did not report the computational time spent by SU on these instances, and therefore we cannot carry out a precise comparison between the two Branch-and-Cut from this point of view.

The first 6 instances and instance n°8 are solved optimally by both algorithms. The instance n°9, instead, is solved to optimality by BC in 1239.4 seconds, while it was not solved by SU within 5000 seconds. Therefore, our algorithm provides a new optimal solution for this set of instances. Both the algorithms certify the infeasibility of instances n°15, 20 and 23. For the remaining 12 instances, BC produces better lower bounds in all cases except one (instance n°21).

Instance				SU		BC		
id	n	m	p	LB	UB	LB	UB	Time
1	50	200	199	708		708		0.2
2	50	200	398	770		770		0.5
3	50	200	597	917		917		1.8
4	50	200	995	1324		1324		7.4
5	100	300	448	4041		4041		4.6
6	100	300	897	5658		5658		178.5
7	100	300	1344	6621.2	-	**6635.4**	-	5010.0
8	100	500	1247	4275		4275		11.5
9	100	500	2495	5951.4	6006	**5997**		1239.4
10	100	500	3741	6510.8	9440	**6707.8**	**8049**	5010.1
11	100	500	6237	7568.7	-	**7729.3**	-	5010.0
12	100	500	12474	9816.9	-	**10560.2**	-	5010.0
13	200	600	1797	13072.9	14707	**13171.2**	**14086**	5010.0
14	200	600	3594	17532.7	-	**17595.0**	-	5010.0
15	200	600	5391	Infeasible		Infeasible		16.4
16	200	800	3196	20744.2	21852	**20941.5**	**21553**	5010.1
17	200	800	6392	26361.3	-	**26526.7**	-	5010.1
18	200	800	9588	29443.6	-	**30634.2**	-	5010.0
19	200	800	15980	33345.1	-	**36900.2**	-	5010.0
20	300	800	3196	Infeasible		Infeasible		2911.1
21	300	1000	4995	51451.3	-	51398.4	-	5010.0
22	300	1000	9990	60907.8	-	**61878.9**	-	5010.0
23	300	1000	14985	Infeasible		Infeasible		1820.0

Table 3.5: Comparison between the solution values of SU and BC algorithms.

With respect to the subset of instances that are not solved by BC and SU, both algorithms found upper bounds in the same 3 cases (instances n° 10, 13 and 16), and those found by BC are always better. It is worth noting the percentage gap value between the upper and the lower bounds in these cases. This value is computed as $100 \times \frac{UB-LB}{UB}$. On the instance n°10, the percentage gap is equal to 31.03% for SU and 16.66% for BC. On the instance n°13, it is equal to 11.11% for SU and 6.49% for BC. Finally, for the instance n°16, it is equal to 5.07% for SU and 2.84% for BC. That is, the percentage gap of BC for these instances is about half of the percentage gap of SU.

Regarding the performance, all the instances optimally solved by BC required less

than 12 seconds, except for the instance n°9 for which, as previously mentioned, about 1240 seconds were spent. To certify the infeasibility, BC required around 16 seconds on the instance n°15, 2911 seconds on the instance n°20 and 1820 seconds on the instance n°23.

In order to further investigate the effectiveness and performance of BC, we generated a new set of benchmark instances. The number of nodes n in this new set ranges from 25 to 100, with incremental steps of size 25. The number of edges m is assigned according to the following density values: 0.2, 0.3, 0.4. A random integer weight chosen in the interval [10,30] is assigned to each edge. That is, a graph with density d has $m = dn(n-1)/2$ edges. This means that our instances are much denser than the previous ones, in which the highest density value is about 0.16.

Given m edges, we can generate at most $\binom{m}{2} = m(m-1)/2$ conflict pairs. The number of conflict pairs associated to each instance is equal to 1%, 4% and 7% of $m(m-1)/2$. We generated 5 different instances for each combination of parameters n, m and p. Thus, in total we generated 180 new instances. The combinations of these parameters allow us to determine which of them affects most the BC performances. It is also worth noting that the new instances were generated ensuring their feasibility, therefore there are no unfeasible instances as in the previous set.

We show the results on this new dataset in Tables 3.6, 3.7, 3.8 and 3.9. The meaning of *id*, n, m and p are the same as for Table 3.5. Under the s columns we report the value of a seed parameter that was used to generate different instances with the same parameter values. The column MST indicates the value of the minimum spanning tree without taking into account the conflicts set P. The last three columns report, as in Table 3.5, the results of our approach (lower bound (LB) and upper bound (UB), or a value in between when an optimum is found) and the computational times in seconds.

We can see that all instances with $n = 25$ (Table 3.6) are solved to optimality within 0.5 seconds, with 38 out of 45 of them requiring under 0.1 seconds. We can, however, start noticing a trend with respect to how parameters affect the complexity of the instances. Indeed, the 4 instances that required the most time to be solved all correspond to cases in which the number of conflict pairs is the highest, with respect to the other instances with the same number of edges. These cases correspond to instances n°36 ($m = 60, p = 124$), n°50 ($m = 90, p = 281$), n°65 and 66 ($m = 120, p = 500$), that are solved in 0.3, 0.5, 0.5 and 0.4 seconds, respectively.

The trend is confirmed for instances of all sizes. For $n = 50$ (Table 3.7) we can observe that all instances with p equal or less than the 4% of $\binom{m}{2}$ are again solved optimally, with computational times growing up to 6.3 seconds for $m = 245$, 7.5 seconds for $m = 367$ and 13.8 seconds for $m = 490$. When p grows to the 7% of the maximum number of conflicts, the related instances result to be considerably more difficult to solve, since we reach a certified optimal solution only for 2 out of 15 of them, namely instances n°80 and 83, solved in 1938.69 and 25.7 seconds, respectively. In the other 13 cases, the gap between the returned lower and upper bounds are between 2% and 8% for $m = 245$, between 3% and 8% for $m = 367$ and between 3% and 7% for $m = 490$.

When $n = 75$ (Table 3.8) we are able to find optimal solutions for all the 15 instances with p equal to the 1% of $\binom{m}{2}$. Computational times in these cases grow up to 23.9 seconds once (instance n°132) and are lower than 5.5 seconds for the remaining 14 instances. None of the remaining 30 instances is solved to optimality. When $p = 4\%$ of $\binom{m}{2}$, we were always able to find both an upper and a lower bound, with gaps between 1% and 6% for $m = 555$, between 1% and 5% for $m = 832$ and between 2% and 6% for $m = 1110$. It can be noticed that in 3 out of 5 cases for $m = 832$ as well as in all 5 cases with $m = 1110$ the computational times are lower than the time limit, as in these cases it was the memory limit to be reached first. The instances with the p equal to the 7% of $\binom{m}{2}$ are again the hardest, since we were able to identify a lower bound only for one of them (instance n°156). Even in this case, the gap between upper and lower bound is considerably high, being equal to 26%. In 13 out of 15 cases we reached the memory limit.

Finally, we consider the results for instances with $n = 100$, reported in Table 3.9. Again, all instances with $p = 1\%$ of $\binom{m}{2}$ could be solved to optimality, within 71.8 seconds for $m = 990$, 249.6 seconds for $m = 1485$ and 214.4 seconds for $m = 1980$. None of the instances with $p = 4\%$ of $\binom{m}{2}$ was solved to optimality, and we were able to identify a lower bound for each of them except one (instance n°164). The gaps between lower and upper bounds are between 12% and 20% for $m = 990$, between 11% and 17% for $m = 1485$ and between 10% and 13% for $m = 1980$. The time limit was always reached for the 5 instances with the smallest number of edges, while the memory limit was always reached in the remaining 10 cases. Finally, when $p = 7\%$ of $\binom{m}{2}$ we were never able to find an upper bound. The memory limit was reached for 6 of these instances, while the time limit was reached in the other 9 cases. With respect to

the MST column, we note that for no instance of our new dataset the optimal solution coincides with this trivial lower bound.

To conclude we can note that, predictably, the factor that most affects the BC performances is the ratio between the number of edges and the number of conflict pairs. Indeed, as p grows with respect to m, it becomes more difficult to find feasible solutions. Between instances with the same number of nodes, increasing the number of edges while keeping constant this ratio have in many cases either marginal or unnoticeable effect on the performances. While increasing the number of nodes leads to harder instances, even the largest ones (with up to 100 nodes and 1980 edges) with the fewest number of conflict pairs could be solved to optimality within about 4 minutes.

Valid inequalities performance analysis In this section we evaluate the impact of our valid inequalities on the effectiveness and performance of BC. To this end, we compare BC with a "basic" Branch-and-Cut composed by constraints (3.3)-(3.6) and (3.9). We will refer to this approach with the name Basic from now on. We carry on this comparison on the larger and more diverse set of instances that we introduced in this chapter. We recall that these instances are guaranteed to be feasible. In order to better assess the effectiveness of our new valid inequalities, we did not provide a starting solution for these tests. Furthermore, a 5000 seconds time limit was considered also for these tests.

The results of this comparison are reported in Tables 3.10, 3.11, 3.12 and 3.13 . The first column reports the instance id. The following eight columns report, for each of the two approaches, the lower bound (*LB*), the upper bound (*UB*), the computational time in seconds (*Time*) and the percentage gap (*Gap*) between the UB and LB values, returned by CPLEX. As in the previous tables, whenever an optimal solution is found, it is reported between the LB and UB columns.

No relevant information can be derived from the smallest instances with $n = 25$, since they are all optimally solved by both models in less than a second (see Table 3.10). The results in Table 3.11 (referring to $n = 50$) show that Basic and BC do not solve to optimality the same subset of 14 instances. On these instances, the gap value of BC is smaller than the one of Basic in 11 out of 14 cases. In 6 of these cases (instances n°80, 81, 82, 95, 96 and 98) the gap difference is higher than 4%, while it is

Instance						BC		
id	n	m	p	s	MST	LB	UB	Time
24	25	60	18	1	336	347		0.0
25	25	60	18	7	384	389		0.0
26	25	60	18	13	350	353		0.0
27	25	60	18	19	345	346		0.0
28	25	60	18	25	330	336		0.0
29	25	60	71	31	343	381		0.0
30	25	60	71	37	334	390		0.1
31	25	60	71	43	346	372		0.0
32	25	60	71	49	328	357		0.0
33	25	60	71	55	379	406		0.0
34	25	60	124	61	321	385		0.0
35	25	60	124	67	363	432		0.0
36	25	60	124	73	335	458		0.3
37	25	60	124	79	338	400		0.0
38	25	60	124	85	340	420		0.0
39	25	90	41	91	299	311		0.0
40	25	90	41	97	305	306		0.0
41	25	90	41	103	293	299		0.0
42	25	90	41	109	294	297		0.0
43	25	90	41	115	314	318		0.0
44	25	90	161	121	280	305		0.0
45	25	90	161	127	316	339		0.0
46	25	90	161	133	310	344		0.0
47	25	90	161	139	296	329		0.0
48	25	90	161	145	301	326		0.0
49	25	90	281	151	317	349		0.0
50	25	90	281	157	321	385		0.5
51	25	90	281	163	288	335		0.0
52	25	90	281	169	295	348		0.1
53	25	90	281	175	295	357		0.0
54	25	120	72	181	281	282		0.0
55	25	120	72	187	287	294		0.0
56	25	120	72	193	276	284		0.0
57	25	120	72	199	277	281		0.0
58	25	120	72	205	290	292		0.0
59	25	120	286	211	300	321		0.0
60	25	120	286	217	296	317		0.0
61	25	120	286	223	271	284		0.0
62	25	120	286	229	296	311		0.0
63	25	120	286	235	283	290		0.0
64	25	120	500	241	290	329		0.1
65	25	120	500	247	285	339		0.5
66	25	120	500	253	306	368		0.4
67	25	120	500	259	277	311		0.0
68	25	120	500	265	275	321		0.0

Table 3.6: Computational results of BC on new instances: $n = 25$

Instance						BC		
id	n	m	p	s	MST	LB	UB	Time
69	50	245	299	271	607	619		0.0
70	50	245	299	277	592	604		0.0
71	50	245	299	283	620	634		0.0
72	50	245	299	289	600	616		0.1
73	50	245	299	295	579	595		0.0
74	50	245	1196	301	590	678		1.4
75	50	245	1196	307	587	681		3.2
76	50	245	1196	313	606	709		6.3
77	50	245	1196	319	575	639		1.5
78	50	245	1196	325	577	681		3.8
79	50	245	2093	331	567	791.20	833	5010.1
80	50	245	2093	337	604	835		1938.7
81	50	245	2093	343	577	773.23	840	5010.1
82	50	245	2093	349	598	820.02	836	5010.1
83	50	245	2093	355	594	769		25.7
84	50	367	672	361	562	570		0.1
85	50	367	672	367	545	561		1.4
86	50	367	672	373	555	573		0.0
87	50	367	672	379	553	560		0.0
88	50	367	672	385	543	549		0.5
89	50	367	2687	391	551	612		7.5
90	50	367	2687	397	546	615		6.6
91	50	367	2687	403	528	587		3.0
92	50	367	2687	409	549	634		7.3
93	50	367	2687	415	587	643		3.2
94	50	367	4702	421	558	701.26	726	5010.1
95	50	367	4702	427	555	719.45	770	5010.0
96	50	367	4702	433	571	723.89	786	5010.0
97	50	367	4702	439	541	669.84	711	5010.0
98	50	367	4702	445	599	737.31	764	5010.0
99	50	490	1199	451	537	548		0.1
100	50	490	1199	457	525	530		0.5
101	50	490	1199	463	543	549		0.0
102	50	490	1199	469	532	540		0.2
103	50	490	1199	475	534	540		0.0
104	50	490	4793	481	546	594		7.8
105	50	490	4793	487	529	579		13.8
106	50	490	4793	493	539	589		3.0
107	50	490	4793	499	528	577		7.5
108	50	490	4793	505	529	592		6.0
109	50	490	8387	511	534	631.43	678	5010.0
110	50	490	8387	517	528	626.72	651	5010.0
111	50	490	8387	523	539	658.38	689	5010.0
112	50	490	8387	529	541	662.22	682	5010.1
113	50	490	8387	535	542	641.31	674	5010.0

Table 3.7: Computational results of BC on new instances: $n = 50$.

Instance						BC		
id	n	m	p	s	MST	LB	UB	Time
114	75	555	1538	541	837	868		0.7
115	75	555	1538	547	842	871		3.0
116	75	555	1538	553	805	838		0.3
117	75	555	1538	559	833	855		4.4
118	75	555	1538	565	830	857		4.1
119	75	555	6150	571	851	1023.72	1047	5010.0
120	75	555	6150	577	839	1008.82	1069	5010.2
121	75	555	6150	583	814	987.31	1040	5010.1
122	75	555	6150	589	828	985.64	998	5010.1
123	75	555	6150	595	825	962.55	994	5010.1
124	75	555	10762	601	817	1054.25	-	4647.3
125	75	555	10762	607	860	1069.51	-	3483.6
126	75	555	10762	613	815	1040.97	-	5010.0
127	75	555	10762	619	798	1006.30	-	5010.0
128	75	555	10762	625	838	1046.43	-	3208.1
129	75	832	3457	631	779	798		4.8
130	75	832	3457	637	795	821		1.1
131	75	832	3457	643	797	816		4.0
132	75	832	3457	649	802	820		23.9
133	75	832	3457	655	789	815		1.4
134	75	832	13828	661	782	873.83	903	3463.3
135	75	832	13828	667	805	901.81	953	4443.7
136	75	832	13828	673	780	873.67	892	5010.0
137	75	832	13828	679	786	885.57	915	4570.8
138	75	832	13828	685	792	886.87	896	5010.0
139	75	832	24199	691	805	949.55	-	1305.3
140	75	832	24199	697	788	907.80	-	1161.2
141	75	832	24199	703	789	910.00	-	953.0
142	75	832	24199	709	800	943.98	-	1224.2
143	75	832	24199	715	803	956.31	-	962.3
144	75	1110	6155	721	777	787		2.1
145	75	1110	6155	727	764	785		5.4
146	75	1110	6155	733	766	783		0.0
147	75	1110	6155	739	774	784		3.3
148	75	1110	6155	745	778	797		5.7
149	75	1110	24620	751	777	846.69	867	4067.5
150	75	1110	24620	757	760	829.23	851	4573.5
151	75	1110	24620	763	767	841.54	892	2189.8
152	75	1110	24620	769	767	841.62	864	2866.0
153	75	1110	24620	775	763	835.04	882	1783.3
154	75	1110	43085	781	769	868.72	-	1049.9
155	75	1110	43085	787	768	853.45	-	1350.7
156	75	1110	43085	793	779	884.67	1194	1522.3
157	75	1110	43085	799	762	853.00	-	1466.3
158	75	1110	43085	805	766	853.98	-	1461.4

Table 3.8: Computational results of BC on new instances: $n = 75$.

Instance						BC		
id	n	m	p	s	MST	LB	UB	Time
159	100	990	4896	811	1070	1119		43.7
160	100	990	4896	817	1094	1137		11.8
161	100	990	4896	823	1076	1113		71.8
162	100	990	4896	829	1066	1110		48.6
163	100	990	4896	835	1044	1090		35.8
164	100	990	19583	841	1086	1249.38	-	5010.0
165	100	990	19583	847	1066	1225.76	1491	5010.0
166	100	990	19583	853	1063	1215.00	1510	5010.0
167	100	990	19583	859	1093	1264.17	1441	5010.2
168	100	990	19583	865	1080	1257.27	1560	5010.1
169	100	990	34269	871	1047	1262.00	-	3006.9
170	100	990	34269	877	1051	1290.68	-	3371.9
171	100	990	34269	883	1091	1318.54	-	3684.2
172	100	990	34269	889	1064	1282.38	-	3939.1
173	100	990	34269	895	1061	1304.45	-	3103.7
174	100	1485	11019	901	1049	1079		248.6
175	100	1485	11019	907	1029	1056		113.4
176	100	1485	11019	913	1034	1059		47.9
177	100	1485	11019	919	1024	1046		195.2
178	100	1485	11019	925	1040	1072		249.6
179	100	1485	44075	931	1040	1143.95	1374	3018.3
180	100	1485	44075	937	1030	1143.61	1291	2144.6
181	100	1485	44075	943	1028	1137.62	1344	3075.6
182	100	1485	44075	949	1028	1136.90	1286	3523.3
183	100	1485	44075	955	1028	1134.63	1370	2954.2
184	100	1485	77131	961	1019	1164.44	-	4773.8
185	100	1485	77131	967	1028	1168.20	-	5010.0
186	100	1485	77131	973	1031	1180.02	-	5010.0
187	100	1485	77131	979	1040	1183.53	-	5010.0
188	100	1485	77131	985	1030	1159.25	-	5010.0
189	100	1980	19593	991	1011	1031		214.4
190	100	1980	19593	997	1015	1036		42.8
191	100	1980	19593	1003	1007	1024		21.8
192	100	1980	19593	1009	1011	1025		27.4
193	100	1980	19593	1015	1010	1028		151.0
194	100	1980	78369	1021	1015	1096.83	1234	1938.3
195	100	1980	78369	1027	1005	1065.64	1187	2160.3
196	100	1980	78369	1033	1015	1087.39	1213	3595.6
197	100	1980	78369	1039	1009	1081.26	1221	2411.8
198	100	1980	78369	1045	1008	1084.09	1245	2385.6
199	100	1980	137145	1051	1004	1098.61	-	5010.1
200	100	1980	137145	1057	1010	1126.27	-	5010.1
201	100	1980	137145	1063	1012	1111.27	-	5010.1
202	100	1980	137145	1069	1024	1114.58	-	5010.1
203	100	1980	137145	1075	1014	1114.07	-	5010.2

Table 3.9: Computational results of BC on new instances: $n = 100$.

higher than 6% in 3 of these cases. In the 3 cases in which the gap of Basic is smaller, the difference is always lower than 2% (see instances n°97, 110 and 112). With respect to the computational time performances, we note that BC is faster than Basic for 29 out of the 31 instances that are solved to optimality by both approaches. In the two cases in which Basic is faster the difference is negligible, being smaller than 0.2 seconds. On the other hand, when BC is faster, the difference is greater than 3 seconds in 6 cases, and greater than 2 seconds in 10 cases. The peak for both algorithms is instance n°83, that is solved in 139 seconds by Basic and in 46.3 seconds by BC.

From the results of Table 3.12 ($n = 75$), we note that 15 instances are solved to optimality by both models. In these cases, BC is always faster, often by a significant margin. Indeed, BC requires up to one fourth of the time required by Basic in 7 out of 15 cases, and up to one third in 9 cases. Moreover, both approaches find upper and lower bounds for the same 15 instances. For BC, the gap between upper and lower bounds is smaller than 5% in 11 cases, while it is smaller than 5% only in 4 cases for Basic. Finally, no feasible solution is found by either of the two approaches for the remaining 15 instances. For these instances, BC finds better lower bounds than Basic in 12 cases.

Finally, we comment the results for the instances with $n = 100$ (Table 3.13). Both algorithms solve to optimality 15 instances. In these cases, BC requires less computational time than Basic 13 times. We observe in particular that BC is about 500 times faster for the instances n°189, and about 25 times faster for the instance n°193. In 5 cases, BC is at least twice faster than Basic. On the other hand, Basic is significantly faster only once (instance n°176).For the remaining 30 instances, Basic is able to find feasible solutions only twice (instances n°194 and 198), while BC finds feasible solutions in 5 additional cases (instances n°180, 183, 195, 196 and 197). Finally, considering the 23 instances for which both algorithms do not find feasible solutions, BC finds better lower bounds than Basic 17 times.

Overall, BC outperforms Basic significantly, being able to find more feasible solutions, and having in most cases either faster convergence times or better solution gaps when a time limit is reached.

id	Basic				BC			
	LB	UB	Time	Gap	LB	UB	Time	Gap
24	347		0.0	0.00%	347		0.0	0.00%
25	389		0.0	0.00%	389		0.0	0.00%
26	353		0.0	0.00%	353		0.0	0.00%
27	346		0.0	0.00%	346		0.0	0.00%
28	336		0.0	0.00%	336		0.0	0.00%
29	381		0.0	0.00%	381		0.0	0.00%
30	390		0.2	0.00%	390		0.1	0.00%
31	372		0.0	0.00%	372		0.0	0.00%
32	357		0.0	0.00%	357		0.0	0.00%
33	406		0.0	0.00%	406		0.0	0.00%
34	385		0.0	0.00%	385		0.0	0.00%
35	432		0.0	0.00%	432		0.0	0.00%
36	458		0.3	0.00%	458		0.3	0.00%
37	400		0.0	0.00%	400		0.0	0.00%
38	420		0.2	0.00%	420		0.0	0.00%
39	311		0.0	0.00%	311		0.0	0.00%
40	306		0.0	0.00%	306		0.0	0.00%
41	299		0.0	0.00%	299		0.0	0.00%
42	297		0.0	0.00%	297		0.0	0.00%
43	318		0.0	0.00%	318		0.0	0.00%
44	305		0.0	0.00%	305		0.0	0.00%
45	339		0.0	0.00%	339		0.0	0.00%
46	344		0.0	0.00%	344		0.0	0.00%
47	329		0.2	0.00%	329		0.1	0.00%
48	326		0.1	0.00%	326		0.1	0.00%
49	349		0.1	0.00%	349		0.0	0.00%
50	385		0.5	0.00%	385		0.4	0.00%
51	335		0.3	0.00%	335		0.0	0.00%
52	348		0.2	0.00%	348		0.2	0.00%
53	357		0.1	0.00%	357		0.1	0.00%
54	282		0.0	0.00%	282		0.0	0.00%
55	294		0.0	0.00%	294		0.0	0.00%
56	284		0.0	0.00%	284		0.0	0.00%
57	281		0.0	0.00%	281		0.0	0.00%
58	292		0.0	0.00%	292		0.0	0.00%
59	321		0.0	0.00%	321		0.1	0.00%
60	317		0.0	0.00%	317		0.0	0.00%
61	284		0.0	0.00%	284		0.0	0.00%
62	311		0.0	0.00%	311		0.0	0.00%
63	290		0.0	0.00%	290		0.0	0.00%
64	329		0.6	0.00%	329		0.3	0.00%
65	339		0.7	0.00%	339		0.6	0.00%
66	368		0.3	0.00%	368		0.4	0.00%
67	311		0.3	0.00%	311		0.3	0.00%
68	321		0.1	0.00%	321		0.0	0.00%

Table 3.10: Computational comparison between Basic and BC algorithms: $n = 25$.

id	Basic				BC			
	LB	UB	Time	Gap	LB	UB	Time	Gap
69	619		1.2	0.00%	619		0.1	0.00%
70	604		0.2	0.00%	604		0.1	0.00%
71	634		0.1	0.00%	634		0.0	0.00%
72	616		0.5	0.00%	616		0.2	0.00%
73	595		0.1	0.00%	595		0.0	0.00%
74	678		3.1	0.00%	678		1.9	0.00%
75	681		3.9	0.00%	681		2.4	0.00%
76	709		7.5	0.00%	709		6.1	0.00%
77	639		1.5	0.00%	639		1.3	0.00%
78	681		8.0	0.00%	681		5.3	0.00%
79	773.43	826	5010.1	6.36%	781.81	820	5010.0	4.66%
80	806.36	870	5010.0	7.31%	832.56	842	5010.0	1.12%
81	759.95	869	5010.0	12.55%	769.48	811	5010.1	5.12%
82	798.78	857	5010.0	6.79%	820.00	836	5010.0	1.91%
83	769		139.0	0.00%	769		46.3	0.00%
84	570		0.2	0.00%	570		0.2	0.00%
85	561		2.1	0.00%	561		2.1	0.00%
86	573		0.0	0.00%	573		0.2	0.00%
87	560		0.0	0.00%	560		0.0	0.00%
88	549		1.5	0.00%	549		1.5	0.00%
89	612		6.6	0.00%	612		10.5	0.00%
90	615		9.5	0.00%	615		7.6	0.00%
91	587		4.3	0.00%	587		3.9	0.00%
92	634		13.7	0.00%	634		7.9	0.00%
93	643		6.0	0.00%	643		2.7	0.00%
94	691.99	741	5010.0	6.61%	696.65	744	5010.0	6.36%
95	708.06	797	5010.0	11.16%	719.42	753	5010.0	4.46%
96	716.14	838	5010.0	14.54%	718.56	797	5010.1	9.84%
97	668.85	711	5010.1	5.93%	664.13	721	5010.0	7.89%
98	726.68	810	5010.0	10.29%	731.33	777	5010.0	5.88%
99	548		3.3	0.00%	548		0.9	0.00%
100	530		1.2	0.00%	530		0.7	0.00%
101	549		3.1	0.00%	549		0.4	0.00%
102	540		5.8	0.00%	540		0.3	0.00%
103	540		0.3	0.00%	540		0.0	0.00%
104	594		7.5	0.00%	594		6.5	0.00%
105	579		17.3	0.00%	579		14.7	0.00%
106	589		7.6	0.00%	589		4.6	0.00%
107	577		9.7	0.00%	577		6.6	0.00%
108	592		8.8	0.00%	592		7.6	0.00%
109	626.81	684	5010.1	8.36%	632.20	666	5010.0	5.08%
110	619.41	658	5010.0	5.86%	621.49	663	5010.0	6.26%
111	654.44	683	5010.0	4.18%	653.81	678	5010.0	3.57%
112	654.48	700	5010.1	6.50%	655.34	710	5010.1	7.70%
113	640.66	677	5010.0	5.37%	644.17	657	5010.0	1.95%

Table 3.11: Computational comparison between Basic and BC algorithms: $n = 50$.

id	Basic				BC			
	LB	UB	Time	Gap	LB	UB	Time	Gap
114	868		10.5	0.00%	868		0.9	0.00%
115	871		11.5	0.00%	871		5.7	0.00%
116	838		5.2	0.00%	838		0.3	0.00%
117	855		12.6	0.00%	855		7.1	0.00%
118	857		7.4	0.00%	857		4.3	0.00%
119	1012.44	1107	5010.0	8.54%	1016.73	1059	5010.1	3.99%
120	1001.68	1089	5010.1	8.02%	1003.86	1114	5010.0	9.89%
121	980.65	1057	5010.1	7.22%	984.46	1084	5010.0	9.18%
122	975.79	1013	5010.0	3.67%	979.31	1017	5010.2	3.71%
123	953.45	1060	5010.0	10.05%	960.47	1003	5010.1	4.24%
124	1046.74	-	5010.0	-	1054.48	-	5010.0	-
125	1064.37	-	5010.0	-	1070.88	-	5010.0	-
126	1033.18	-	5010.0	-	1040.94	-	5010.0	-
127	1006.13	-	5010.0	-	1006.35	-	5010.0	-
128	1047.47	-	5010.0	-	1048.05	-	5010.0	-
129	798		51.3	0.00%	798		36.7	0.00%
130	821		50.5	0.00%	821		1.0	0.00%
131	816		25.1	0.00%	816		15.2	0.00%
132	820		33.8	0.00%	820		23.3	0.00%
133	815		40.3	0.00%	815		8.9	0.00%
134	871.54	916	5010.2	4.85%	875.65	891	5010.0	1.72%
135	897.83	969	5010.1	7.34%	899.49	947	5010.1	5.02%
136	867.48	943	5010.1	8.01%	869.94	905	5010.1	3.87%
137	879.01	952	5010.1	7.67%	879.68	920	5010.1	4.38%
138	883.89	899	5010.0	1.68%	884.78	896	5010.0	1.25%
139	951.91	-	5010.1	-	955.79	-	4332.2	-
140	912.91	-	5010.1	-	913.03	-	3362.6	-
141	913.16	-	5010.1	-	915.04	-	2839.6	-
142	950.36	-	5010.1	-	949.84	-	3795.0	-
143	958.86	-	5010.1	-	960.83	-	2776.8	-
144	787		55.9	0.00%	787		17.7	0.00%
145	785		84.3	0.00%	785		10.0	0.00%
146	783		68.1	0.00%	783		1.3	0.00%
147	784		69.5	0.00%	784		12.0	0.00%
148	797		69.9	0.00%	797		22.4	0.00%
149	847.08	855	5010.0	0.93%	846.49	857	5010.0	1.23%
150	825.59	897	5010.1	7.96%	827.87	860	5010.2	3.74%
151	839.97	902	5010.0	6.88%	840.73	901	5010.2	6.69%
152	838.72	894	5010.2	6.18%	840.80	877	5010.2	4.13%
153	835.07	880	5010.2	5.11%	837.77	868	5010.1	3.48%
154	873.46	-	3332.9	-	870.27	-	2357.1	-
155	854.24	-	3116.3	-	857.00	-	3106.8	-
156	883.54	-	3801.8	-	885.94	-	3443.8	-
157	856.67	-	3946.8	-	856.22	-	3448.3	-
158	857.41	-	3938.4	-	859.11	-	3146.2	-

Table 3.12: Computational comparison between Basic and BC algorithms: $n = 75$.

id	Basic				BC			
	LB	UB	Time	Gap	LB	UB	Time	Gap
159	1119		117.3	0.00%	1119		88.8	0.00%
160	1137		58.3	0.00%	1137		12.3	0.00%
161	1113		126.0	0.00%	1113		66.7	0.00%
162	1110		91.9	0.00%	1110		71.0	0.00%
163	1090		93.6	0.00%	1090		20.4	0.00%
164	1247.59	-	5010.0	-	1249.38	-	5010.0	-
165	1216.97	-	5010.1	-	1217.29	-	5010.0	-
166	1206.80	-	5010.0	-	1211.03	-	5010.0	-
167	1256.52	-	5010.1	-	1258.81	-	5010.0	-
168	1250.44	-	5010.0	-	1253.76	-	5010.0	-
169	1254.25	-	5010.0	-	1264.77	-	5010.0	-
170	1288.38	-	5010.0	-	1295.60	-	5010.0	-
171	1309.62	-	5010.0	-	1323.71	-	5010.0	-
172	1274.36	-	5010.0	-	1283.59	-	5010.0	-
173	1298.12	-	5010.0	-	1309.11	-	5010.0	-
174	1079		270.0	0.00%	1079		282.7	0.00%
175	1056		195.6	0.00%	1056		141.2	0.00%
176	1059		8.1	0.00%	1059		142.8	0.00%
177	1046		218.2	0.00%	1046		117.3	0.00%
178	1072		367.5	0.00%	1072		249.0	0.00%
179	1141.12	-	5010.1	-	1141.83	-	5010.1	-
180	1141.56	-	5010.1	-	1141.95	1801	5010.2	36.59%
181	1133.96	-	5010.2	-	1134.27	-	5010.1	-
182	1133.23	-	5010.2	-	1135.19	-	5010.1	-
183	1131.49	-	5010.2	-	1132.51	1691	5010.1	33.03%
184	1163.15	-	5010.0	-	1164.44	-	5010.0	-
185	1160.56	-	5010.0	-	1168.20	-	5010.0	-
186	1187.83	-	5010.0	-	1180.02	-	5010.0	-
187	1183.69	-	5010.0	-	1183.53	-	5010.0	-
188	1164.47	-	5010.0	-	1159.25	-	5010.0	-
189	1031		2293.5	0.00%	1031		4.3	0.00%
190	1036		549.6	0.00%	1036		256.5	0.00%
191	1024		601.8	0.00%	1024		319.8	0.00%
192	1025		582.9	0.00%	1025		472.9	0.00%
193	1028		750.4	0.00%	1028		30.1	0.00%
194	1096.39	1550	5010.4	29.27%	1097.00	1521	5010.1	27.88%
195	1062.91	-	5010.2	-	1064.18	1540	5010.1	30.90%
196	1084.42	-	5010.1	-	1086.79	1437	5010.2	24.37%
197	1080.75	-	5010.1	-	1083.45	1506	5010.2	28.06%
198	1083.57	1597	5010.3	32.15%	1083.75	1684	5010.1	35.64%
199	1100.57	-	5010.1	-	1098.61	-	5010.1	-
200	1125.85	-	5010.1	-	1126.27	-	5010.1	-
201	1110.66	-	5010.1	-	1111.27	-	5010.1	-
202	1118.69	-	5010.1	-	1114.58	-	5010.1	-
203	1114.14	-	5010.1	-	1114.07	-	5010.2	-

Table 3.13: Computational comparison between Basic and BC algorithms: $n = 100$.

Chapter 4

A flow formulation for the Close-enough arc routing problem

4.1 Introduction

The close-enough arc routing problem is a generalization of the classic arc routing problem and it has many interesting real-life applications. In this chapter, we propose some techniques to reduce the size of the input graph and a new effective mixed integer programming formulation for the problem. Our experiments on directed graphs show the effectiveness of our reduction techniques. Computational results obtained by comparing our MIP model with the existing exact methods show that our algorithm is really effective in practice.

4.2 Problem definition

The Close-Enough Arc Routing Problem (CEARP) is a generalization of the Rural Postman Problem (RPP). Let $G = (V, A, M)$ be a directed graph with a set of vertices V, a set of arcs A, and a set of targets M located on arcs. An arc $a \in A$ covers a target $m \in M$ iff the target is either on the arc or within a predetermined distance (radius) from the arc. Let $N = \{(m,a) | m \in M, a \in A\}$ be a set containing the couple (target m, arc a) if and only if the arc a covers the target m, and let c_{ij} be the cost associated with arc $a = (i, j) \in A$. Finally, let $v_0 \in V$ indicate the depot node. The *CEARP* consists of finding

a minimum cost tour starting and ending at the depot node v_0, traversing a subset of arcs such that all the targets in M are covered. The CEARP problem was introduced by Drexl [20, 21], he proved that the problem is NP-hard, and he proposed a branch-and-cut algorithm. Shuttleworth et al. [62] proposed four heuristics to solve instances with approximately 9000 arcs. A mixed integer programming (MIP) formulation for the problem was introduced by Há et al. [37]; the same authors presented a new IP formulation in [38]. Ávila et al. [2] proposed a branch-and-cut algorithm which they compared with the model presented in [38], providing good computational results. In [48], Lum et al. propose some techniques to partition a graph in order to simplify its structure.

The remainder of this chapter is organized as follows. In Section 4.3, we present our new MIP formulation. In Section 4.4, we show the computational results of our approach, comparing them with recent results proposed in [2] and [38] and, finally, in Section **??**, we present our conclusions.

4.3 Flow formulation

In this section, we propose a very effective mathematical programming formulation for the CEARP based on an efficient graph reduction procedure. Moreover, we show how, using any vertex cover computed on the input graph, we can take advantage of its features to identify a set of important vertices for the routing problems.

4.3.1 Graph reduction

In order to improve the resolution process, we prove some properties of the problem and we provide some definitions that will help us to reduce the size of input instances.

we prove some problem's properties useful to reduce the size of input instances using some idea presented in [48]. For this purpose, we give the following definitions:

Definition 4.3.1 *We say that the edge $e \in E$ cover the meter $m \in M$, if $(m,e) \in A$.*

Definition 4.3.2 *For each $m \in M$, $C(m) = \{a \in A | (m,a) \in N\}$ is the set of all the arcs that cover the target m.*

It is possible to reduce the number of targets, taking into consideration the following property:

Property 4.3.1 *Let $m_1, m_2 \in M$ be a couple of targets, the target m_2 is redundant if $C(m_1) \subseteq C(m_2)$.*

proof 4.3.1 *Let $G' = (V, A, M')$ be the graph where $M' = M \setminus \{m_2\}$. Each tour T' in G' covering each target in M' contains at last one arc $a \in C(m_1)$ and then the tour T' is also a feasible tour for the graph $G = (V, A, M)$ (see Fig. 4.1).*

In the following, we give two definitions to characterize the set of vertices and the set of arcs that are necessary in every feasible solution.

Definition 4.3.3 *The set $\hat{V} = \{v \in V \mid \exists m \in M$ such that $v = i$ or $v = j$, $\forall (i, j) \in C(m)\}$ is the set of necessary vertices.*

Definition 4.3.4 *The set $\hat{A} = \{a \in A \mid \exists m \in M \ : \ C(m) = \{a\}\}$ is the set of necessary arcs.*

Finally, by using the following two properties, we can try to reduce the size of the input instances.

Property 4.3.2 *The target $m \in M$ is redundant if $\exists\, a \in \hat{A}$ such that $a \in C(m)$.*

Property 4.3.3 *The target $m \in M$ is redundant if $\exists\, (i, j) \in C(m)$ such that $i \in \hat{V}$ or $j \in \hat{V}$.*

4.3.2 The Vertex Cover

In this section, to strengthen the MIP model, we will use some information obtained from solving a vertex cover problem related to the directed graph $G = (V, A, M)$ defining our problem. We consider the graph $G' = (V, E, M)$ obtained from G by transforming each directed arc into an undirected edge and leaving in E only edges associated with arcs of G that cover at least one target in M. On G', we solve a vertex cover problem getting a subset (VC) of its vertices such that each edge of G' is incident to at least

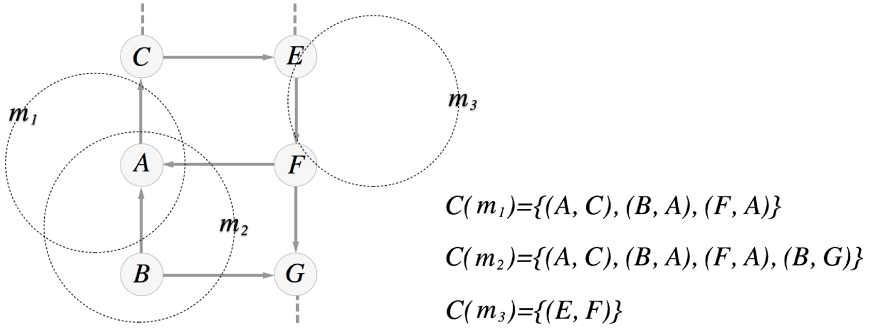


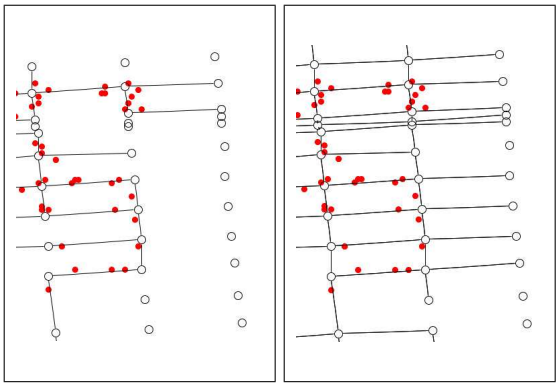
Figure 4.1: Redundant target m_2. Necessary vertex A. Necessary arc (E,F).

one vertex of VC. In Fig 4.2(a), we show a portion of a real street network. The red dots represent the targets that we need to cover and the edges are those in which at least one target is located. In Fig 4.2(b) are drawn all edges in which it is possible to cover at least one target (we construct G' considering only these edges). By computing the set $VC \subseteq V$ as a vertex cover obtained using the edges of Fig 4.2(b), we get a set of vertices VC from which we can efficiently check if a set of arcs on G, necessary to cover all the targets in M, is connected for each feasible solution. Obviously the vertex cover with the minimum cardinality would be the best possible set VC, but, in many cases, it is sufficient to compute just a feasible solution using an heuristic procedure [14] or a metaheuristic approach like tabu search [5] or a genetic algorithm [13]. Heuristically, we can compute VC using a simple two-step procedure. In step 1, we insert in VC all the vertices in $\hat{V}$. In step 2, we add to VC all the other vertices necessary to get a feasible solution. In Fig 4.2(c), we report, in black, the vertices in the set $\hat{V}$ and, in green, the vertices in the set $VC \setminus \hat{V}$. If we look at the edges as bi-directed arcs, the red edges represent a feasible solution T for the CEARP. Indeed each edge in T is incident to at least a green or a black vertex.

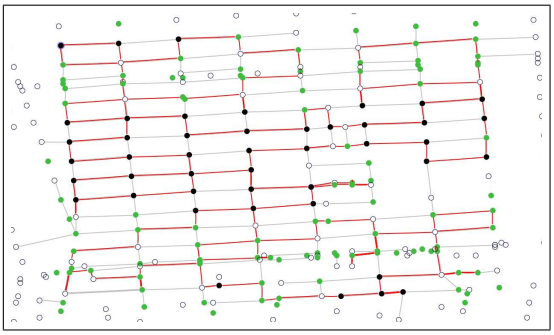
4.3.3 MIP model

In this section, we describe our MIP model based on a flow formulation. From now on, the depot will be vertex 0. In order to proceed with the description of the model, we define the variables that will be used. For each arc $(i, j) \in A$, we have:

- $x_{ij} \in Z_0^+$ is the number of times the arc (i, j) is traversed.



(a) (b)



(c)

Figure 4.2: (a) Edges in which there is at least one target (red dot). (b) All edges in which it is possible to read at least one target. (c) A feasible solution.

- $f_{ij} \in R_0^+$ are the flow variables associated with arc (i, j).

The model can be formulated as follows:

$$\text{Minimize} \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{4.1}$$

$$\sum_{a=(i,j)\in A|(m,a)\in N} x_{ij} \geq 1 \qquad \forall\, m \in M \tag{4.2}$$

$$x_{ij} \geq 1 \qquad \forall\, (i,j) \in \hat{A} \tag{4.3}$$

$$\sum_{j\in V|(i,j)\in A} x_{ij} - \sum_{j\in V|(j,i)\in A} x_{ji} = 0 \qquad \forall\, i \in V \tag{4.4}$$

$$\sum_{(0,j)\in A} f_{0j} \geq 1 \tag{4.5}$$

$$\sum_{j\in V|(i,j)\in A} f_{ij} - \sum_{j\in V|(j,i)\in A} f_{ji} = 0 \qquad \forall\, i \in V \setminus (VC \cup \{0\}) \tag{4.6}$$

$$\sum_{j\in V|(i,j)\in A} f_{ij} - \sum_{j\in V|(j,i)\in A} f_{ji} = 1 \qquad \forall\, i \in \hat{V} \setminus \{0\} \tag{4.7}$$

$$\sum_{j\in V|(i,j)\in A} f_{ij} - \sum_{j\in V|(j,i)\in A} f_{ji} = \sum_{j\in V|(i,j)\in A} x_{ij} \qquad \forall\, i \in VC \setminus (\hat{V} \cup \{0\}) \tag{4.8}$$

$$f_{ij} \leq M x_{ij} \qquad \forall\, (i,j) \in A \tag{4.9}$$

$$x_{ij} \in Z_0^+ \qquad \forall\, (i,j) \in A \tag{4.10}$$

$$f_{ij} \in R_0^+ \qquad \forall\, (i,j) \in A \tag{4.11}$$

The set of constraints (4.2) ensure that each target is covered at least from one arc of the solution and constraints (4.3) ensure that each necessary arc is in the solution. The set of constraints (4.4) ensure that for each node, the number of selected arcs in its forward star corresponds to the number of selected arcs in its backward star. The constraint (4.5) ensures outcoming flow from the depot. The set of constraints (4.6) set to 0 the amount of in-flow for all the vertices not necessary to guarantee the connection of the final solution. Constraints (4.7) set to 1 the amount of in-flow for all the vertices that we know will be traversed in any feasible solution. Constraints (4.8) defined for each vertex that could be used to ensure the connection of the solution, set an in-flow equal to the time that the vertex is traversed in the solution. Constraints (4.9) ensure

that we can have flow only on the arcs used in the solution.

4.4 Computational results

In this section, we present computational results obtained by using the MIP model. Our experiments were performed on a OSX 10.9 operating system, 16 GB of RAM and a quad-core processor Intel I7 running at 2.6 GHz. The MIP model was coded in Java and solved using IBM ILOG CPLEX 12.5.

The computational tests are performed on the set of benchmark instances presented in [37]. We compared our MIP model with the exact approaches proposed in [2, 38]. In Table 4.1 there are two sets of instances. In the first set, we have $|V| = 500$ and $|A| = 1500$ while, in the second one, we have $|V| = 500$ and $|A| = 1000$. For each set, we have four different numbers of targets $(\frac{1}{2}|A|,\ |A|,\ 5|A|,\ 10|A|)$.

Table 4.1 shows that our algorithm solves to optimality 39 instances. For these two sets, our approach is competitive with the approach of Avilá et al. and outperforms the results of Há et al.. Table 4.2 shows that in terms of computational times our MIP model outperforms the previous approaches. In terms of numbers of solved instances, our approach is always better than Há et.al. [38]. For the sparse instances $(|A| = 1000)$, our model outperforms the results of Ávila et al. [2] with respect to computation times. For the dense instances $(|A| = 1500)$, Ávila et al. are able to find the optimal solution in 20 instances. Our approach finds the optimal solution in 19 instances; we can certify optimality for 14 instances and for the remaining 6 instances, our gap is always less than the 1%.

	Há et al.	Avilá et al.	Cerrone et al.		Há et al.	Avilá et al.	Cerrone et al.
1500_0_0.5	105431.4	**104892.6**	**104892.6**	1000_0_0.5	76333.0	**76033.0**	**76033.0**
1500_1_0.5	97335.3	**96766.0**	96791.6	1000_1_0.5	84337.2	**84237.2**	**84237.2**
1500_2_0.5	112429.6	**112102.1**	**112102.1**	1000_2_0.5	*89353.5	**89653.5**	**89653.5**
1500_3_0.5	95378.1	**94897.8**	**94897.8**	1000_3_0.5	76384.4	**75954.9**	**75954.9**
1500_4_0.5	102423.9	**101991.0**	**101991.0**	1000_4_0.5	85397.0	**85097.1**	**85097.1**
1500_0_1	*129459.7	**129570.6**	**129570.6**	1000_0_1	*82387.1	**82687.1**	**82687.1**
1500_1_1	123423.0	**123123.0**	**123123.0**	1000_1_1	*89396.5	**89896.5**	**89896.5**
1500_2_1	**133418.3**	**133418.3**	**133418.3**	1000_2_1	98351.8	**98051.8**	**98051.8**
1500_3_1	116458.8	**115943.8**	**115943.8**	1000_3_1	**82344.2**	**82344.2**	**82344.2**
1500_4_1	117403.4	**116721.5**	**116721.5**	1000_4_1	*91315.6	**91915.6**	**91915.6**
1500_0_5	162497.8	**162097.8**	**162097.8**	1000_0_5	100495.3	**100395.4**	**100395.4**
1500_1_5	*160492.7	**160792.8**	**160792.8**	1000_1_5	109418.5	**109318.5**	**109318.5**
1500_2_5	177442.4	**177242.4**	**177242.4**	1000_2_5	114462.3	**114362.3**	**114362.3**
1500_3_5	*151452.9	**151852.9**	**151852.9**	1000_3_5	*103470.9	**103791.0**	**103791.0**
1500_4_5	*161433.4	**161833.4**	**161833.4**	1000_4_5	*112425.0	**112625.0**	**112625.0**
1500_0_10	*174404.1	**174504.1**	**174504.1**	1000_0_10	*110427.9	**110528.0**	**110528.0**
1500_1_10	**173404.5**	**173404.5**	**173404.5**	1000_1_10	*113494.1	**113694.2**	**113694.2**
1500_2_10	185430.8	**185330.8**	**185330.8**	1000_2_10	*123433.9	**126660.0**	**126660.0**
1500_3_10	162471.7	**162071.7**	**162071.7**	1000_3_10	115481.4	**115281.4**	**115281.4**
1500_4_10	*168434.3	**168734.3**	**168734.3**	1000_4_10	128463.2	**128163.3**	**128163.3**

Table 4.1: In both subtables, the first column shows the name of the instance, the remaining three columns show the objective function values. The star (*) is associated with outliers (maybe depending on a different parsing of the instances). Optimal solutions are in bold.

	Há et al.		Avilá et al.		Cerrone et al.	
	Opt found	Time	Opt found	Time	Opt found	Time
1500_0.5	0	7202.9	5	830.5	1	874.0
1500_1	2	4499.9	5	1235.5	3	433.5
1500_5	5	154.5	5	49.2	5	14.6
1500_10	5	205.9	5	50.1	5	9.9
1000_0.5	3	4155.6	5	245.7	5	47.7
1000_1	4	2447.8	5	88.6	5	13.3
1000_5	5	315.1	5	28.3	5	6.0
1000_10	5	82.3	5	20.3	5	4.4

Table 4.2: For the three models compared in this section, we show the running times (seconds) and the number of certified optimal solutions. Each row of the table shows the average value on five instances.

Chapter 5

Properties, formulation and a two-level metaheuristic for the all-colors shortest path problem

Given an undirected graph in which each vertex is assigned to a color, in the all-colors shortest path problem we look for a minimum cost shortest path spanning all different colors. The problem is known to be NP-Hard and hard to approximate. In this chapter we propose some new properties, as well as a compact representation for feasible solutions. Furthermore, we present a novel mathematical formulation and a metaheuristic approach based on these ideas. Computational results show the effectiveness of our approach with respect to previous contributions.

5.1 Introduction

The *All-Colors Shortest Path* (ACSP) is a combinatorial optimization problem, first introduced in [3]. The problem is defined on undirected graphs, in which a numerical attribute (weight) is associated to each edge, while a logical attribute (called *color*, or *label*) is given for each vertex. Therefore the different colors, appearing in the graph, partition the set of vertices into disjoint subsets. The aim of ACSP is to find the shortest, possibly non-simple path, spanning each color of the graph; that is, each path composing a feasible solution needs to visit at least a vertex belonging to each color.

As mentioned, the optimal solution of the problem could correspond to a non-simple path, meaning that a vertex may be visited more than once. As an example, consider the graph illustrated in Figure 5.1. The value reported on each edge represents its cost (for instance, the weight of $\{v_1, v_2\}$ is 4), while the c_i label next to each vertex denotes its color. A (simple) path reaching every color is, for instance, $[v_1, v_4, v_2, v_3, v_6, v_5, v_7]$, whose cost is 14. Despite visiting twice vertices v_4 and v_5, and visiting once the additional vertex v_8, the all-colors shortest path is instead $[v_1, v_4, v_2, v_4, v_5, v_3, v_5, v_6, v_8, v_7]$. Indeed, in this case, the cost of the path is 10.

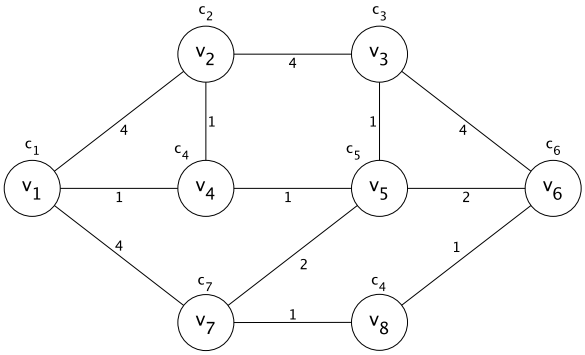


Figure 5.1: Example graph with 8 vertices, 12 edges and 7 colors

To the best of our knowledge, the only previous contribution in the literature regarding ACSP is [3]. In this work, after introducing the problem, the authors show it to be NP-Hard and inapproximable to a constant factor. They then present an integer linear programming flow-based formulation, as well as three heuristic and three metaheuristic algorithms. The heuristic algorithms are iterative rounding methods based on LP relaxations of the mathematical formulation, while the metaheuristics include a simulated annealing (SA), an ant colony optimization method (ACO) and a genetic algorithm (GA).

It is important to note that the definition of the problem given in [3] does not match exactly ours. Indeed, in the previous work the path is required to start from a predefined source vertex. We call this variant the *All-Colors Shortest Path with Starting Vertex* (ACSP-SV). In a further variant, one may consider as set of candidate starting vertices those assigned to a given color (*All-Colors Shortest Path with Starting Color*, or ACSP-SC). In this chapter, we mainly focus on the ACSP problem variant, that is, we do not

make any assumption about the starting vertex; however, all the approaches that we present are easily adaptable to solve ACSP-SV and ACSP-SC. Indeed, we also present a comparison among our heuristic and model and those proposed in [3] for ACSP-SV in Section 5.5. In Section 2, we also show ACSP and ACSP-SV to be computationally equivalent.

In this chapter, we propose some properties, a novel formulation and a metaheuristic algorithm to solve ACSP. The metaheuristic that we propose is a Variable Neighborhood Search (VNS), and is based on the concept of two-level solutions. That is, for each problem solution we consider a *high-level*, abstract representation of it, corresponding to the order in which colors are encountered, in addition to the *low-level* (actual) one. As will be shown, by jointly operating on the two levels we obtained a fast and effective algorithm.

The rest of the chapter is organized as follows. In Section 5.2 we define the problem formally, and present some of its properties. In Section 5.3 we present a mathematical formulation for the problem, while our VNS algorithm is discussed in Section 5.4. Section 5.5 presents our computational results.

5.2 Problem definition and properties

Let $G = (V,E,C)$ be an undirected, connected and vertex labeled graph, where $V = \{v_1,\ldots,v_n\}$ is the set of vertices, $E = \{e_1,\ldots,e_m\}$ is the set of edges and $C = \{c_1,\ldots,c_k\}$ is a set of labels (or colors), with $|C| \leq |V|$. Moreover, let $\omega : E \to \mathbb{R}^+$ be a function assigning a positive weight to each edge, and $\gamma : V \to C$ be a function assigning a color to each vertex. We denote by V_c the subset of vertices of V having the color c, that is $V_c = \{v \in V : \gamma(v) = c\}$. We use the notation $p = [v_1^p, v_2^p, \ldots, v_h^p]$ to denote a path p of G; that is, for each $i \in \{1,\ldots h-1\}$, $v_i^p \in V$, $v_{i+1}^p \in V$ and $\{v_i^p, v_{i+1}^p\} \in E$.

The aim of ACSP is to find a path $p = [v_1^p, v_2^p, \ldots, v_h^p]$ such that **i)** all colors are reached at least once, that is, $\forall c_j \in C\ \exists v_i^p : \gamma(v_i^p) = c_j$; **ii)** the overall weight of the path $\omega(p) = \sum_{i=1}^{h-1} \omega(\{v_i^p, v_{i+1}^p\})$ is minimized.

In ACSP-SV, the starting vertex of any given feasible solution must be a predefined source vertex $v_{src} \in V$. In ACSP-SC, the starting vertex must be chosen among those associated with a predefined color $c_{src} \in C$.

Clearly, the problem is correctly defined only if $|C| \geq 2$; indeed, if $|C| = 1$ selecting

any vertex brings to a trivial optimal solution with value 0. It is also straightforward to observe that no feasible solution exists if a given color $c_i \in C$ is not assigned to any vertex in V.

We now discuss some ACSP properties. In [3], the authors proved that the following results hold for ACSP-SV:

Theorem 5.2.1 *ACSP-SV is NP-Hard.*

Theorem 5.2.2 *ACSP-SV is inapproximable to a constant factor.*

Proposition 5.2.3 *Let $[v_{src}, \dots, v_h]$ be the optimal solution for an ACSP-SV instance. The path does not traverse any edge $\{v_i, v_j\}$ more than once in the same direction.*

The proof provided in [3] for Proposition 5.2.3 is directly applicable to ACSP as well, while the ones for Theorems 1 and 2 are adaptable with trivial modifications. Furthermore, in the following we show the existence of polynomial-time reductions from each of the two problems to the other.

We start from the reduction from ACSP to ACSP-SV. Let $G = (V, E, C)$ be an input graph for ACSP. Considering a new vertex v' and a new color c', such that $\gamma(v') = c'$, we build a new graph $G' = (V', E', C')$, where $V' = V \cup \{v'\}$, $E' = E \cup \{\{v', v_i\}\ \forall v_i \in V\}$ and $C' = C \cup \{c'\}$. Each edge $\{v_i, v_j\} \in E$ has the same weight in both G and G', and each node $v_i \in V$ is assigned the same color in the two graphs. Furthermore, the weight of each edge incident to v' in G' is equal to $2|E|\omega_{max} + 1$, where ω_{max} is $\max(\{\omega(\{v_i, v_j\}) : \{v_i, v_j\} \in E\})$.

We first need to show a preliminary result.

Lemma 5.2.4 *Let $p' = [v', v_1^p \dots, v_h^p]$ be an optimal solution for ACSP-SV in the above described graph G', with source vertex v'. The path p' visits v' exactly once.*

proof 5.2.1 *By contradiction, let us suppose that p' visits v' more than once, i.e. there exists at least a $k = 1, \dots, h$ such that $v_k^p = v'$. Since $\omega(\{v', v_i\}) = 2|E|\omega_{max} + 1$, $\forall v_i \in V$, $\omega(p') \geq 4|E|\omega_{max} + 2$. Now, let $q = [v_1^q \dots, v_k^q]$ be an optimal ACSP solution in G. From Proposition 5.2.3, we know that $\omega(q) \leq 2|E|\omega_{max}$. By adding the vertex v' at the beginning of q, we obtain a new path $q' = [v', v_1^q \dots, v_k^q]$ that is a ACSP-SV feasible solution with $\omega(q') \leq 4|E|\omega_{max} + 1$. This means that $\omega(q') < \omega(p')$, contradicting the hypothesis.*

We are now ready to prove the reduction.

Proposition 5.2.5 *The path $p = [v_1^p \dots, v_h^p]$ is an optimal solution for ACSP in G if and only if $p' = [v', v_1^p \dots, v_h^p]$ is an optimal solution for ACSP-SV in G', with source vertex v'.*

proof 5.2.2 $\boxed{\Longrightarrow}$ *Let us first assume that p is optimal for ACSP in G and, by contradiction, that p' is not optimal for ACSP-SV in G' with source v'. We note that, however, p' is surely a feasible solution for the latter problem. Let $q' = [v', v_1^q \dots, v_k^q]$ be the optimal one; it follows that $\omega(q') < \omega(p')$. Furthermore, by Lemma 5.2.4, we know that $q = [v_1^q \dots, v_k^q]$ obtained by removing $\{v', v_1^q\}$ from q', does not contain v' and is therefore a feasible ACSP solution in G. Since $\omega(\{v', v_1^p\}) = \omega(\{v', v_1^q\})$, then $\omega(q) < \omega(p)$, which contradicts the hypothesis on the optimality of p.*

$\boxed{\Longleftarrow}$ *Now, let us assume that p' is optimal and that p is not. Again from Lemma 5.2.4, we know that p is feasible for ACSP in G. Let $q = [v_1^q \dots, v_k^q]$ be an optimal ACSP solution in G; therefore, $\omega(q) < \omega(p)$. We obtain that $q' = [v', v_1^q \dots, v_k^q]$ is feasible for ACSP-SV in G' with source v', and $\omega(q') < \omega(p')$, which is again a contradiction.*

Let us now illustrate the reduction from ACSP-SV to ACSP. Let $G = (V, E, C)$ be an input graph for ACSP-SV, with source vertex $v_{src} \in V$. Considering a new vertex v' and a new color c', such that $\gamma(v') = c'$, we build a new graph $G' = (V', E', C')$, where $V' = V \cup \{v'\}$, $E' = E \cup \{\{v', v_{src}\}\}$ and $C' = C \cup \{c'\}$. Edges $\{v_i, v_j\} \in E$ have the same weight in both G and G', and nodes $v_i \in V$ have the same color in the two graphs. Moreover, the weight of $\{v', v_{src}\}$ is equal to $2|E|\omega_{max} + 1$ (again, ω_{max} is $\max(\{\omega(\{v_i, v_j\}) : \{v_i, v_j\} \in E\})$).

We will first present two preliminary results (Lemmas 5.2.6 and 5.2.7), and then prove the reduction (Proposition 5.2.8).

Lemma 5.2.6 *Any optimal solution for ACSP in the above described graph G' visits v' exactly once, and v' is one of the two endpoints of the path.*

proof 5.2.3 *Let us assume that $p = [v_{src} \dots, v_h^p]$ is an optimal ACSP-SV solution in G. From Proposition 5.2.3, $\omega(p) \leq 2|E|\omega_{max}$. We note that $p' = [v', v_{src} \dots, v_h^p]$ is feasible for ACSP in G', since it visits all colors in C' using edges contained in E', and $\omega(p') \leq 4|E|\omega_{max} + 1$.*

Any feasible ACSP solution in G' needs to visit v', in order to reach color c'. However, by construction, if such a solution visits v' multiple times or contains v' as an internal node, it crosses the edge $\{v', v_{src}\}$ more than once. Therefore, such a solution has a weight that is strictly greater than the one of p', and is not optimal.

Lemma 5.2.7 *Given any feasible ACSP solution $[v_1^p \dots, v_h^p]$, the reverse path $[v_h^p \dots, v_1^p]$ is also a feasible solution and has identical weight.*

Since ACSP is defined on undirected graphs, it is straightforward to see that Lemma 5.2.7 holds.

Proposition 5.2.8 *The path $p = [v_{src} \dots, v_h^p]$ is an optimal solution for ACSP-SV in G (with source v_{src}) if and only if $p' = [v', v_{src} \dots, v_h^p]$ is an optimal solution for ACSP in G'.*

proof 5.2.4 *First of all we note that, from Lemmas 5.2.6 and 5.2.7, whenever considering an optimal ACSP solution in G', without loss of generality we can consider v' to be its starting vertex. We also note that, by construction, its adjacent vertex will always be v_{src}.*
$\boxed{\Longrightarrow}$ *Let us first assume that p is optimal for ACSP-SV in G with source v_{src}, and that p' is not optimal for ACSP in G'. Obviously, p' is a feasible solution for the latter problem. Let $q' = [v', v_{src} \dots, v_k^q]$ be the optimal one, and let us call q its subpath $[v_{src} \dots, v_k^q]$. From Lemma 5.2.6, q does not contain v'. Since q visits all colors in C, it is a feasible ACSP-SV solution in G with source v_{src}. Moreover, $\omega(q') < \omega(p')$, therefore $\omega(q) < \omega(p)$ in G, which contradicts the hypothesis on the optimality of p.*
$\boxed{\Longleftarrow}$ *Let $p' = [v', v_{src} \dots, v_h^p]$ be optimal for ACSP in G'; again from Lemma 5.2.6, its subpath $p = [v_{src} \dots, v_h^p]$ is feasible for ACSP-SV in G with source v_{src}. If we suppose that p is not optimal, then there must exist a feasible solution $q = [v_{src} \dots, v_k^q]$ such that $\omega(q) < \omega(p)$. However, in this case, $q' = [v', v_{src} \dots, v_k^q]$ is a feasible ACSP solution in G' and its objective function is better than the one of p', which is again a contradiction.*

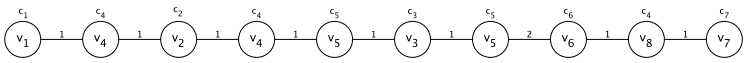
We now report some additional properties that we found for ACSP, and that we used in both our mathematical model and VNS.

Proposition 5.2.9 *Let $p^* = [v_1^{p^*}, \dots, v_h^{p^*}]$ be an optimal solution for the ACSP problem. Then **i)** $\gamma(v_h^{p^*})$ must be different from $\gamma(v_i^{p^*})$ $\forall i = \{1, .., h-1\}$, and **ii)** $\gamma(v_1^{p^*})$ must be different from $\gamma(v_i^{p^*})$ $\forall i = \{2, .., h\}$.*

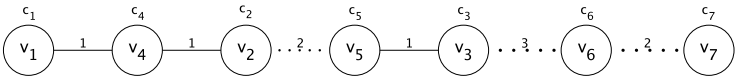
proof 5.2.5 *Let us first prove* ***i)****. It is easy to understand that if* $\gamma(v_i^{p^*}) = \gamma(v_h^{p^*})$ *for some* $i = \{1,..,h-1\}$*, then all colors are visited by the subpath* $p' = [v_1^{p^*}, \ldots, v_{h-1}^{p^*}]$*. But then,* p' *is a feasible solution and it is cheaper than* p^**, contradicting the hypothesis. Given that no assumption is made on* $v_1^{p^*}$*, the sub-property also holds for the ACSP-SV and ACSP-SC problem variants. We can prove* ***ii)*** *analogously. Indeed, if* $\gamma(v_1^{p^*}) = \gamma(v_i^{p^*})$ *for some* $i = \{2,..,h\}$*, it follows that* $p'' = [v_2^{p^*}, \ldots, v_h^{p^*}]$ *is a feasible ACSP solution that is cheaper than* p^**. This sub-property does not necessarily hold for ACSP-SV and ACSP-SC, since the choice of the first endpoint is constrained.*

Before introducing the next property, we present an alternative representation for any feasible solution. Let $p = [v_1^p, v_2^p, \ldots, v_h^p]$ be a feasible path for ACSP, that we also define the low-level representation of the path itself. The high-level representation of p is the path $p' =< v_1^{p'}, v_2^{p'}, \ldots, v_{|C|}^{p'} >$ of $G' = (V, \{V \times V\}, C)$, containing the vertices corresponding to the first occurrence of each color in p. For each consecutive couple of vertices $v_i^{p'}$ and $v_{i+1}^{p'}$, the weight of the related edge is equal to $\omega(\{v_i^{p'}, v_{i+1}^{p'}\})$ if it also belongs to p, or to the sum of the weight of the edges between the two vertices in p otherwise. In the following, we will also use the term high-level (or low-level) solution to refer to a solution in the corresponding representation. Note that we use square and angle brackets to distinguish low-level and high-level solutions, respectively.

In Figure 5.2(a) we show the optimal solution for the example of Figure 5.1, while in Figure 5.2(b) its high-level representation is shown. In this figure, we use dotted lines to highlight edges that substitute subpaths of p. That is, in Figure 5.2(b) edge $\{v_2, v_5\}$ replaces the subpath $[v_2, v_4, v_5]$, $\{v_3, v_6\}$ replaces $[v_3, v_5, v_6]$, and $\{v_6, v_7\}$ replaces $[v_6, v_8, v_7]$.



(a) Low-level (actual) solution



(b) High-level solution

Figure 5.2: Low-level and high-level optimal solution for the instance of Figure 5.1

Clearly, if $p' =< v_1^{p'}, v_2^{p'}, \ldots, v_{|C|}^{p'} >$ is the high-level representation of a path p, they have the same cost and share the same starting vertex. Furthermore, from Proposition 5.2.9, we know that if $v_{|C|}^{p'}$ is not the final vertex of p, we can drop all subsequent vertices from the path and obtain a better feasible solution. In particular, the high-level and the low-level representation of an optimal solution will always share the same endpoints.

We can now present the following result, connected to the concept of high-level solution:

Proposition 5.2.10 *Let $p' =< v_1^{p'}, \ldots, v_i, v_j, \ldots, v_{|C|}^{p'} >$ be a high-level, optimal ACSP solution. Then, the subpath between any couple of consecutive vertices v_i and v_j in the corresponding low-level solution is the shortest path between v_i and v_j in the input graph G.*

proof 5.2.6 *Let v_i and v_j be any couple of consecutive vertices in p', and let $SP(i,j)$ denote their shortest path in G. Let us suppose by contradiction that the low-level representation p of p' contains a subpath between them that is not their shortest path in G; it follows that the cost of this subpath is strictly greater than the one of $SP(i,j)$. Therefore, by replacing this subpath with $SP(i,j)$, we obtain a feasible ACSP solution that is better than p', contradicting the hypothesis.*

5.3 Mathematical model

In this section we introduce a mathematical model for the ACSP, whose solutions will be used to verify the effectiveness of our metaheuristic.

Starting from the graph G, we build a new directed graph $G^d = (V^d, E^d, C^d)$. V^d contains all vertices of V, as well as a dummy source s and a dummy sink t. E^d contains both arcs (v_i, v_j) and (v_j, v_i) for each edge $\{v_i, v_j\} \in E$, and both arcs have the same weight of the original edge. Additionally, E^d contains arcs $\{(s, v_i) : v_i \in V\} \cup \{(v_i, t) : v_i \in V\}$; these arcs have cost zero. We define $\delta^+(v_i, G^d) = \{v_j \in V^d : (v_i, v_j) \in E^d\}$ and $\Delta^+(v_i, G^d) = \{(v_i, v_j) \in E^d\}$. We similarly define $\delta^-(v_i, G^d) = \{v_j \in V^d : (v_j, v_i) \in E^d\}$ and $\Delta^-(v_i, G^d) = \{(v_j, v_i) \in E^d\}$. Finally, C^d contains all colors in C, as well as two additional ones, c_s, c_t; these colors are assigned to s and t, respectively, while all

other nodes have the same color assigned in the two graphs. In the solution, we will look for a path from s to t crossing at least a vertex for each color in C.

The decision variables are the following:

- x_i: binary variable equal to 1 if vertex $v_i \in V$ belongs to the solution, and 0 otherwise.
- y_{ij}: binary variable equal to 1 if arc $(v_i, v_j) \in E^d$ belongs to the solution, and 0 otherwise. These variables are binary since no arc of E^d will be crossed more than once in the solution found by the model. By effect of Proposition 5.2.3, this does not compromise its optimality.

The mathematical model is the following:

$$\min \sum_{(v_i,v_j)\in E^d} \omega(v_i,v_j) y_{ij} \tag{5.1}$$

$$\sum_{v_i \in V_c} x_i \geq 1 \qquad c \in C \tag{5.2}$$

$$\sum_{v_k \in \delta^-(v_i,G^d)} y_{ki} \geq x_i \qquad v_i \in V \tag{5.3}$$

$$\sum_{v_k \in \delta^-(v_i,G^d)} y_{ki} = \sum_{v_j \in \delta^+(v_i,G^d)} y_{ij} \qquad v_i \in V \tag{5.4}$$

$$\sum_{v_i \in \delta^+(s,G^d)} y_{si} = 1 \tag{5.5}$$

$$\sum_{i \in \delta^-(t,G^d)} y_{it} = 1 \tag{5.6}$$

$$y_{ij} \leq x_i \qquad (v_i,v_j) \in E^d : v_i \in V \tag{5.7}$$

$$\sum_{(v_i,v_j)\in \Delta^-(v_j,G^d) | v_i \notin S, v_j \in S} y_{ij} \geq x_k \qquad S \subseteq V^d \setminus \{s\}, v_k \in S \tag{5.8}$$

$$x_i \in \{0,1\} \qquad v_i \in V \tag{5.9}$$

$$y_{ij} \in \{0,1\} \qquad (v_i,v_j) \in E^d \tag{5.10}$$

The objective function (5.1) minimizes the cost of the individuated path. Constraints (5.2) ensure that at least a vertex for each color is visited. Constraints (5.3) and (5.4) impose that there is at least an ingoing arc for each visited vertex, and that

the number of ingoing and outgoing arcs is the same for each of them, respectively. Constraints (5.5) and (5.6) state that there must be exactly one edge leaving s and one edge entering t, respectively. Constraints (5.7) state than an arc (v_i, v_j) with $i \in V$ can belong to the solution if v_i belongs to it as well. Constraints (5.8) ensure that all visited vertices are connected to s, and hence that the solution is connected. They are a modified version of the "directed connectivity constraints" and they state that, for each subset $S \subseteq V^d \setminus \{s\}$, if a visited vertex v_k belongs to S then there must be at least one arc entering in S. Finally, constraints (5.9)-(5.10) are variable definitions.

We now present some additional valid inequalities for our model, and briefly discuss how to adapt it to the ACSP-SV and ACSP-SC variants.

5.3.1 Valid inequalities

In order to speed up the resolution of the model, in the following we introduce further constraints to break symmetry and take advantage of some of the properties introduced in the previous section.

- Symmetry often heavily affects the computational time required by integer programming models to find the optimal solution. Unfortunately, in the case of ACSP there is symmetry that must be managed by using additional constraints. Indeed, from Lemma 5.2.7, if $[v_1^p, \dots, v_h^p]$ is a feasible ACSP solution, then $[v_h^p, \dots\dots, v_1^p]$ is feasible as well and has identical cost. These two paths would correspond to two distinct feasible solutions for our model ($[s, v_1^p, \dots, v_h^p, t]$ and $[s, v_h^p, \dots, v_1^p, t]$, respectively).

  We break this symmetry by introducing a new constraint, ensuring that the index of the first endpoint is always lower than the index of the last one. Formally,

$$\sum_{v_i \in V} i y_{si} \leq \sum_{v_j \in V} j y_{jt} \tag{5.11}$$

  This constraint, along with constraints (5.5)-(5.6), produces the desired effect.

- As a consequence of Proposition 5.2.9, we know that in the optimal solution the two endpoints have different colors. Hence, we introduced the following valid

inequalities:

$$\sum_{v_i \in V_c} y_{si} + \sum_{v_j \in V_c} y_{jt} \leq 1 \qquad c \in C \tag{5.12}$$

Our computational tests showed that these constraints improve the LP relaxation value and, in general, they reduce the time needed to find the optimal solution.

- Let v_i be a vertex contained in the optimal path; let c be its color. Again from Proposition 5.2.9, we know that no vertex $v_j \in V_c \setminus \{v_i\}$ can be the final endpoint. This is expressed by the following constraints:

$$\sum_{v_j \in V_c \setminus \{v_i\}} y_{jt} \leq 1 - x_i \qquad c \in C, v_i \in V_c \tag{5.13}$$

Analogous constraints are also valid with respect to the starting endpoint:

$$\sum_{v_j \in V_c \setminus \{v_i\}} y_{sj} \leq 1 - x_i \qquad c \in C, v_i \in V_c \tag{5.14}$$

In the following we refer to the formulation (5.1)-(5.14) as ILP2 formulation.

5.3.2 Adapting the model to ACSP-SC and ACSP-SV

In order to adapt our model for the ACSP-SC and ACSP-SV, it is sufficient to solve the above presented mathematical formulation on differently defined directed graphs. Let us define $G^{d'} = (V^d, E^{d'}, C^d)$ the directed graph for ACSP-SC, and $G^{d''} = (V^d, E^{d''}, C^d)$ the one for ACSP-SV. For each $v_i \in V^d \setminus \{s\}$ and $v_j \in V^d \setminus \{s\}$, both $E^{d'}$ and $E^{d''}$ contain (v_i, v_j) if and only if the arc is contained in E^d, and the arc has the same weight that it has in E^d. Furthermore, $E^{d'}$ contains arcs $\{(s, v_i) : v_i \in V_{c_{src}}\}$, while $E^{d''}$ contains only the arc (s, v_{src}), and these arcs have cost zero.

It is easy to understand that $G^{d'}$ and $G^{d''}$ model the requirements on the first endpoint of the path related to the ACSP-SC and ACSP-SV problems, respectively. Furthermore, given the above mentioned requirements, Constraints (5.11) and (5.14) are

not valid for these variants of the problem.

In Section 5.5, we refer to the ACSP-SV formulation (that is, (5.1)-(5.10),(5.12),(5.13) on graph $G^{d''}$) as ILP2-SV.

5.4 Variable Neighborhood Search

In Section 5.2 we introduced the concept of high-level solutions, whose main advantage is that they represent feasible solutions as fixed-length, simple paths, showing in which order (and with which vertex) each color is first reached. As already discussed, supposing to be able to determine the color visiting sequence of the optimal solution, as well as the correct vertex for each color, finding the optimal solution would be an easy task, since we would just need to connect such vertices by means of shortest paths. Our VNS algorithm is based on this fundamental idea, trying to iteratively improve a current candidate solution as follows:

1. Given a feasible, high-level solution, we look for new solutions by perturbing the color visiting sequence using two classical neighborhood strategies for TSP problems, that is, relocate and 2-opt;
2. Once a new color visiting sequence has been decided in the above step, we determine locally optimal choices for the vertices of the colors involved in the perturbation.

Hence, we determine first the color sequence, and then the actual vertices of new high-level solutions. Once these vertices are chosen for a new high-level solution, it is easy to reconstruct the corresponding low-level one, since these vertices will always be connected by means of shortest paths. In this sense, we say that our approach works on two levels.

We also developed a greedy heuristic based on this fundamental idea, that operates in $|C|-1$ steps and is used to produce the first feasible solution.

Algorithm 1 presents the pseudocode of our metaheuristic approach. After individuating a starting feasible solution using the heuristic algorithm described in Section 5.4.1, we look for improvements by means of a local search that uses relocate neighborhoods As soon as this local search step fails to find an improvement, we apply a new local search using a 2-opt neighborhood strategy. If we manage to improve the

Algorithm 1: VNS pseudocode

```
1  sol ← GreedyInit();
2  bestSol ← sol;
3  for i = 1 to MaxShakes do
4      improvement ← true;
5      while improvement = true do
6          while improvement = true do
7              sol' ← Relocate(sol);
8              if objFunction(sol') < objFunction(sol) then
9                  sol ← sol';
10             else
11                 improvement ← false;
12         sol' ← 2-opt(sol);
13         if objFunction(sol') < objFunction(sol) then
14             sol ← sol';
15             improvement ← true;
16     if objFunction(sol) < objFunction(bestSol) then
17         bestSol ← sol;
18     sol ← Shake(sol);
19 return bestSol;
```

current solution, the algorithm goes back to the relocate local search, otherwise we attempt to escape from the current local optimum by means of a shake operator, and the algorithm iterates. The two local search operators are presented in Section 5.4.2, while the shake operator is discussed in Section 5.4.3. When the 2-opt local search fails and a pre-defined number of shakes has been reached, the algorithm ends and the best solution found is returned.

5.4.1 Initialization algorithm

In a preliminary step, we evaluate the shortest paths among each couple of vertices of G by using the Floyd-Warshall algorithm (see [16]). As well known, the algorithm operates in $O(|V|^3)$ time.

The initialization algorithm then performs $|C|$ steps to produce a feasible ACSP solution. In the first step, a random vertex is chosen as first endpoint. In the i-th

step ($i = 2, \ldots, |C|$) the algorithm chooses, among all vertices whose colors differ from those of the vertices chosen in the previous steps, the one whose shortest path from the vertex chosen in the $(i-1)$-th step has minimum weight.

It is clear that, after the $|C|$-th step, the sequence of chosen vertices is a high-level feasible solution. Since the Floyd-Warshall algorithm allows to reconstruct each shortest path by means of an auxiliary predecessor matrix, we are also able to reconstruct the corresponding low-level solution.

The initialization algorithm is easy to adapt to ACSP-SV and ACSP-SC. Indeed, in the first case the starting vertex is always chosen to be v_{src}, while in the second case it will be a random one among those associated to c_{src}.

5.4.2 Relocate and 2-Opt Local Search

Let $p = < v^p_1, \ldots, v^p_{|C|} >$ be the current high-level solution. Consider the associated color sequence $< c^p_1, \ldots, c^p_{|C|} >$, where $c^p_i = \gamma(v^p_i)$. Our two local search operators work as follows.

- The relocate local search performs $|C|^2 - |C|$ iterations, each building a new neighbor, operating in two steps:
  1. For $i \in \{1, \ldots, |C|\}$, the vertex in the i-th position is removed from p, hence we obtain an incomplete high-level solution $< v^p_1, \ldots, v^p_{i-1}, v^p_{i+1}, v^p_{|C|} >$. If $i = 1$ or $|C|$, the solution is simply truncated to remove the corresponding endpoint. Otherwise, new locally optimal vertex choices are made for colors c^p_{i-1} and c^p_{i+1}, since v^p_{i-1} and v^p_{i+1} may no longer be favorable choices now that they are directly connected in the high-level solution. Consider for instance the case of Figure 5.3(b), in which new vertices must be chosen for c^p_3 and c^p_5 after removing v^p_4 from the solution show in Figure 5.3(a).
  2. For each incomplete solution obtained from the previous step, $|C| - 1$ new complete high-level solutions are obtained by adding a newly chosen vertex with color c^p_i in each position $j \in \{1, \ldots, |C|\}, j \neq i$. For instance, in Figure 5.4(a), color $c^p_i = c^p_4$ is relocated in the second position ($j = 2$).
- In the 2-opt local search, each new neighbor is generated by removing two edges from the current high-level solution and replacing them with two different ones.

Overall, $\frac{|C^2|-|C|}{2}$ neighbors are created, operating as follows:

- For each $i = 1,\ldots,|C|-1$ and $j = i+1,\ldots,|C|$, a neighbor p' whose sequence of colors in its high-level representation is $< c^p_1,\ldots,c^p_{i-1},c^p_j,\ldots,c^p_i,c^p_{j+1},\ldots,c^p_{|C|} >$ is generated, where the color visiting sequence between j and i in p' is the inverse of p. For instance, given the high-level solution p with $|C| = 6$ in Figure 5.5(a), by applying a 2-opt operation corresponding to $i = 3$ and $j = 5$ we obtain a neighbor whose color visiting sequence is shown in Figure 5.5(b), that is, $< c^p_1,c^p_2,c^p_5,c^p_4,c^p_3,c^p_6 >$. The choice for a given color c^p_k in p' will be v^p_k if $k \notin \{i-1,i,j,j+1\}$, while a new choice is required otherwise. That is, we determine new choices for the endpoints of the edges involved in the swap. If $i = 1$, c^p_j becomes the color of the new starting endpoint, and similarly, if $j = |C|$, c^p_i is the new final color.

We now describe how new vertices are chosen when needed for the two steps of each relocate iteration, as well as for each 2-opt iteration. In the following, we define *undecided* the colors for which a new vertex has to be chosen, according to the previously described steps.

The underlying idea is to generate for each of these three steps an auxiliary directed graph $G^a = (V^a,E^a)$, containing all candidate vertices belonging to each undecided color. In G^a, each vertex of an undecided color c_i has an ingoing arc that connects it to the vertex that would precede it in the high-level solution that we are building, as well as an outgoing one to the vertex that would follow it. If c_i is preceded or followed in our solution by another undecided color c_j, its vertices are connected to all vertices of c_j. The weight of each arc in G^a is equal to the cost of the shortest path between its endpoints in the original graph G. Dummy source or destination nodes are considered to handle special cases in which preceding or following nodes do not exist. By looking for shortest paths in G^a, we identify the new vertices for the undecided colors. In more detail:

- **Relocate, Step 1**: Let us assume that v^p_{i-2} and v^p_{i+2} both exist. In this case, V^a is composed of $\{v^p_{i-2},v^p_{i+2}\} \cup V_{c^p_{i-1}} \cup V_{c^p_{i+1}}$. E^a contains edges $(v^p_{i-2},v'),\forall v' \in V_{c^p_{i-1}}$, arcs $(v',v''),\forall v' \in V_{c^p_{i-1}},v'' \in V_{c^p_{i+1}}$ and arcs $(v'',v^p_{i+2}),\forall v'' \in V_{c^p_{i+1}}$. We then use the Dijkstra algorithm to find a shortest path in G^a from v^p_{i-2} and v^p_{i+2}; by construction, this path will cross exactly one vertex with color c^p_{i-1} and one vertex

with color c^p_{i+1}. Figure 5.3(c)-(d) shows graph G^a and the individuated shortest path (edge weights are omitted).

If c^p_{i-1} is the starting color, v^p_{i-2} is substituted by a dummy source, which is connected in G^a to the vertices with color c^p_{i-1} though zero-weighted edges. With an analogous reasoning, we replace v^p_{i+2} with a dummy destination if c^p_{i+1} is the last color reached by the solution.

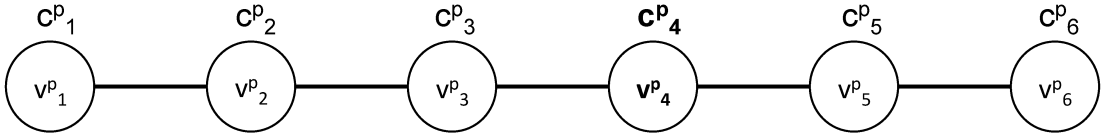
- **Relocate, Step 2**: Let us assume that c^p_i has to be relocated in the j-th position, with $1 < j < |C|$. Let v^{prv} and v^{nxt} be the vertices that will precede and follow the new vertex. In the auxiliary graph, V^a contains $\{v^{prv}, v^{nxt}\} \cup V_{c^p_i}$, while E^a contains arcs (v^{prv}, v') and (v', v^{nxt}) $\forall v' \in V_{c^p_i}$. By looking for a shortest path between v^{prv} and v^{nxt}, we identify a vertex with color c^p_i. Figure 5.4(b)-(d) show these steps and the final high-level representation of the newly built neighbor.

  If $j = 1$, v^{prv} does not exist; the procedure reduces to selecting the vertex in $V_{c^p_i}$ whose shortest path distance from $v^{nxt} = v^p_1$ is minimal. The same holds with respect to $v^{prv} = v^p_{|C|}$ if $j = |C|$.

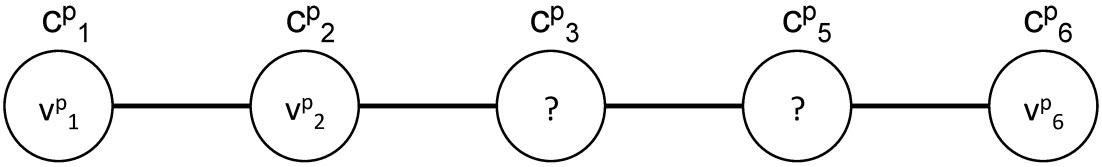
- **2-opt**: In G^a, for each color $c^p_k \in C$, V^a contains v^p_k if $k \notin \{i-1, i, j, j+1\}$, or $V_{c^p_k}$ otherwise. Given any couple of consecutive colors c^p_k and c^p_q in the new solution, E^a will contain an arc from each vertex with color c^p_k to each vertex with color c^p_q in V^a.

  If the first and last endpoint of the path do not belong to undecided colors ($i - 1 > 1$ and $j + 1 < |C|$), we look for a shortest path between them. Otherwise, if $i - 1 \leq 1$, the first endpoint belongs to either color $c^p_{i-1} = c^p_1$ or c^p_j; in both cases, this endpoint is not known. Hence, we consider a dummy source that is connected to each vertex with color c^p_1 (or c^p_j, respectively) with zero-weighted arcs. Analogously, if $j+1 \geq |C|$, the last endpoint of the path has color $c^p_{j+1} = c^p_{|C|}$ or c^p_i. In these cases, we add a dummy destination.

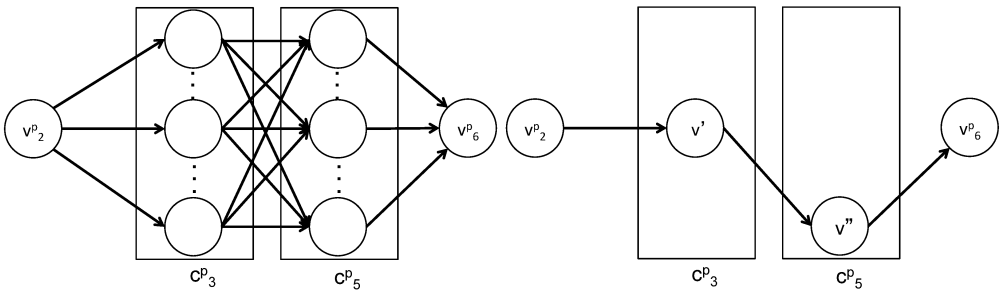
  Figure 5.5(c) shows the G^a auxiliary graph for the considered example; note that since $i - 1 = 2 > 1$ we do not need the dummy source (the first endpoint v^p_1 is known), while the dummy destination is needed ($j = 5 = |C| - 1$). Figures 5.5(d)-(e) show the shortest path found and the related high-level neighbor solution, respectively.



(a) Current high-level solution; vertex with color c_4^p chosen for removal

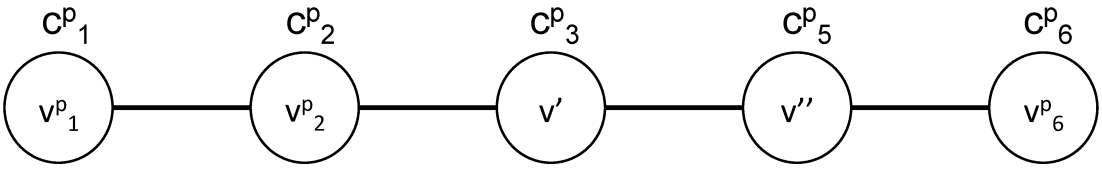


(b) Incomplete solution after the removal; choices needed for c_3^p and c_5^p



(c) Auxiliary graph G^a

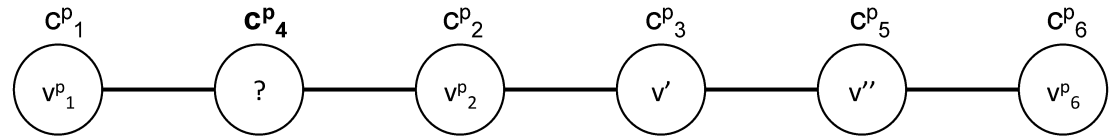
(d) Shortest path identification



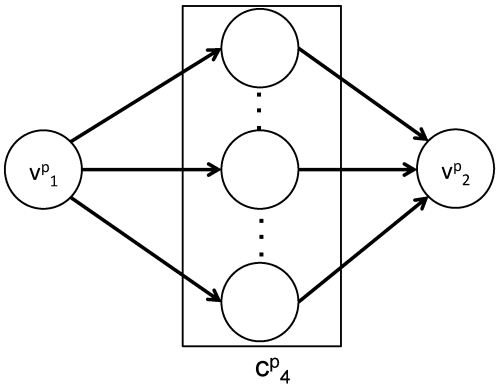
(e) Incomplete solution after the new choices for c_3^p and c_5^p

Figure 5.3: Neighbor construction through Relocate Local Search (step 1, $i = 4$)

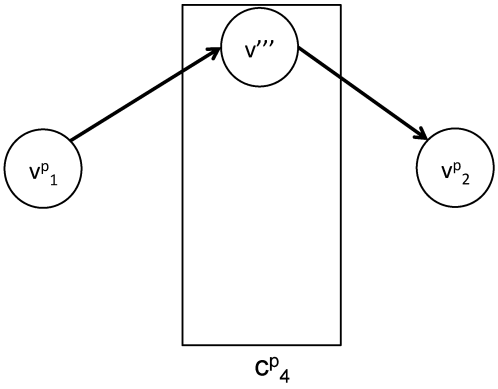
Each relocate local search iteration stores the neighbor solution with minimum cost encountered. Once all neighbors have been generated, if the best one is an improvement with respect to the current solution, a new relocate local search iteration starts. Otherwise, the current solution does not change and the 2-opt local search step is invoked. Similarly to the relocate one, our 2-opt local search explores the whole neighborhood, storing the best solution found. If an improvement with respect to the current solution is found, we go back to the relocate local search; otherwise we attempt to escape from



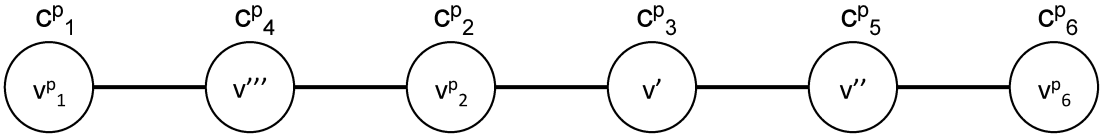
(a) Color c_4^p to be relocated in position 2



(b) Auxiliary graph G^a



(c) Shortest path identification



(d) Complete high-level solution after the new choice for c_4^p

Figure 5.4: Neighbor construction through Relocate Local Search (step 2, $j = 2$)

the current local minimum by performing a shake operation. It is easy to adapt both operators to ACSP-SV or ACSP-SC. For the relocate, the color corresponding to the first endpoint is not removed from the solution in step 1, and no color is relocated in the first position in step 2. Furthermore, with respect to ACSP-SV, when the color in the the second position is removed from the path in step 1, we make sure that the first endpoint is not changed. In order to adapt the 2-opt local search, we avoid the case $i = 1$, since, as described, this case corresponds to a choice of a new color for the first endpoint. Moreover, for ACSP-SV, when $i = 2$ we impose the shortest path in G^a to start from $v_1^p = v_{src}$.

5.4.3 Shake operator

The shake operator performs a number of random perturbations to the current solution, in order to try to escape from the local optimum. In more detail, every time that the shake operator is invoked, it performs $\frac{|C|}{shake_1}$ relocate operations. For each of these operations, the i and j values used in the two steps of the relocate procedure are chosen randomly. Overall, a total of $shake_2$ shake operations are performed, and every $\frac{shake_2}{shake_3}$ invocations of the operator, the current solution is rebuilt from scratch by using the initialization algorithm described in Section 5.4.1. The values chosen for parameters $shake_1$, $shake_2$ and $shake_3$ are reported in Section 5.5.

5.5 Computational results

This section presents the test scenarios and the results obtained during our computational test phase. Our VNS algorithm was coded using the C++ programming language, while the mathematical formulations were implemented and solved using the IBM ILOG CPLEX 12.6.1 solver. All tests were performed in single thread mode on a machine with an Intel Xeon E5-2650 v3 processor running at 2.3 GHz and 128 GB of RAM. With respect to the VNS parameters, after a preliminary tuning phase we chose the values $shake_1 = 3$, $shake_2 = |V|$ and $shake_3 = 5$. For CPLEX, we considered a time limit equal to 3600 seconds. Whenever the solver reaches this threshold, the related solution value is marked with a "*" symbol to highlight that this value is an upper bound of the optimal solution.

The following two subsection contain results for the ACSP-SV and ACSP problems, respectively.

5.5.1 Comparisons on the ACSP-SV problem

In this subsection, we compare the effectiveness and the performance of our formulation and VNS metaheuristic with the formulation and heuristics proposed in [3] for the ACSP-SV problem.

We start by comparing the performance of our ILP2-SV formulation with the formulation proposed in [3], named ILP. Both formulations were implemented by us, and compared on our testing environment.

Instances			ILP2-SV			ILP			GAP	
n	m	k	LP	Obj	Time	LP	Obj	Time	LP	Obj
50	189	10	37.03	40	0.40	26.01	40	10.53	42.39%	0.00%
50	152	20	101.00	101	0.06	79.35	101	3.20	27.29%	0.00%
50	154	25	129.34	132	0.21	106.36	132	6.08	21.60%	0.00%
100	338	25	128.81	138	3.04	94.33	138	86.91	36.55%	0.00%
100	330	40	207.62	220	1.25	148.55	220	19.82	39.76%	0.00%
100	373	50	229.28	233	0.95	192.91	233	35.74	18.85%	0.00%
200	734	50	179.69	223	158.22	119.95	223*	3612.17	49.81%	0.00%
200	746	75	373.51	399	87.47	310.25	399	2013.22	20.39%	0.00%

Table 5.1: Comparison of the ILP2-SV and ILP formulations for ACSP-SV on the instances proposed in [3].

A first comparison between ILP and ILP2-SV was carried out on the dataset of instances proposed in [3], having a number of vertices between 50 and 200 and a number of colors between 10 and 75. The instance files have been provided by the authors.

The results of this comparison are reported in Table 5.1. Under the *Instances* heading, we report the instances characteristics (number of vertices n, number of edges m and number of colors k). The next six columns report the root linear relaxation value (*LP*), the solution value (*Obj*) and the computational time (*Time*), in seconds, for ILP2-SV and ILP, respectively. Finally, under the *GAP* heading, we report the gap percentage between the linear relaxation values and between the solution values, respectively. These gaps are computed by using the formulas $100 \times \frac{LP(ILP2-SV)-LP(ILP)}{LP(ILP)}$ and $100 \times \frac{Obj(ILP2-SV)-Obj(ILP)}{Obj(ILP)}$, respectively.

The results under the GAP heading show that the linear relaxation of ILP2-SV is always better than the one of ILP, with gaps ranging from 18.85% to 49.81%. With respect to solutions quality, we note that ILP2-SV always finds the optimal one, while ILP reaches the time limit once (see the case with 200 nodes and 50 colors), hence it is not able to certify the optimality of the solution found in this case. Regarding computational times, ILP2-SV is always faster than ILP, solving 6 out of 8 instances in less than 4 seconds, and requiring about 158 seconds in the worst case. The computational times of ILP are significantly higher, and worse in all cases. Indeed, as already mentioned, in the worst case it reaches the time limit, on the last instance it requires about 2013 seconds, and on the remaining 6 instances the computational time ranges from 3

to 87 seconds. The results of Table 5.1 clearly show that ILP2-SV outperforms ILP on this dataset.

A second comparison among the two formulations was carried out on a new, larger set of instances. Our instances were generated with respect to 3 parameters: the number of vertices n, the number of edges m and the number of colors k. The vertices are randomly disposed in a square area of size 50x50. The value of n is chosen in the set $\{25, 50, 75, 100, 150\}$. The number of edges m is chosen by using a density d that ranges in the set $\{0.2, 0.3, 0.4, 0.5\}$ ($m = \frac{n(n-1)}{2} \times d$). Finally, the number of colors k belongs to the set $\{\lceil 0.1n \rceil, \lceil 0.2n \rceil, \lceil 0.3n \rceil, \lceil 0.4n \rceil\}$. We generated 5 instances for each combination of parameters, discarding the case $n = 25$, $k = \lceil 0.1n \rceil = 3$, which resulted to be particularly trivial and in which it would not make sense to define the 2-opt operator (these instances were also used to test our VNS, as will be discussed in Section 5.5.2). Therefore, our dataset is composed in total of 76 different scenarios and 380 individual instances. For these tests, the node indexed with 0 was also assumed to be the ACSP-SV source vertex. Our instances are available online[1].

Table 5.2 contains the results of the comparison between ILP2-SV and ILP on this new set of instances. Table headings have the same meaning that they have for Table 5.1, with the addition of column m which reports the value of the additional instance parameter. However, in this case we report average values for each scenario.

We can note that the linear relaxation values of ILP2-SV are again always better than the ones of ILP, with a percentage gap that ranges from about 1.5% to over 100%. In 56 out of 76 scenarios, the percentage gap is greater than 20%. As a consequence, we observe a remarkable difference between the effectiveness of ILP2-SV and ILP. Indeed, ILP2-SV reaches the time limit without finding the optimal solution only once (150 nodes, 4470 edges, 30 colors). On the other hand, ILP reaches the time limit 24 times, and the first failures occur on the instances with 75 nodes. The solution percentage gap is lower than -5% in 14 out these 24 cases, and it decreases down to about -35%. We note in particular that ILP never finds the optimal solution for the instances with $n = 150$. With respect to performances, ILP2-SV is most of the times an order of magnitude faster than ILP. We can note that the scenarios with up to 100 nodes are optimally solved by ILP2-SV within about 3.5 minutes. On the same scenarios, ILP reaches the time limit 8 times. In the 15 scenarios with 150 nodes solved to optimality

[1] http://www.dipmat2.unisa.it/people/carrabs/www/DataSet/ACSP_Instances.zip

Instances			ILP2-SV			ILP			GAP	
n	m	k	LP	Obj	Time	LP	Obj	Time	LP	Obj
25	60	5	67.97	72.40	0.08	47.31	72.40	0.30	43.69%	0.00%
25	90	5	57.33	61.40	0.07	39.36	61.40	0.49	45.65%	0.00%
25	120	5	39.66	47.40	0.04	32.80	47.40	0.33	20.90%	0.00%
25	150	5	39.19	46.00	0.10	29.96	46.00	0.52	30.79%	0.00%
25	60	8	116.64	117.20	0.11	86.73	117.20	1.04	34.48%	0.00%
25	90	8	98.22	100.20	0.03	74.97	100.20	1.40	31.02%	0.00%
25	120	8	82.26	89.20	0.13	68.63	89.20	1.66	19.86%	0.00%
25	150	8	80.25	85.20	0.10	66.46	85.20	1.64	20.76%	0.00%
25	60	10	149.80	157.20	0.01	137.37	157.20	0.44	9.05%	0.00%
25	90	10	118.16	137.40	0.04	107.68	137.40	1.55	9.73%	0.00%
25	120	10	109.14	110.00	0.07	99.76	110.00	1.09	9.41%	0.00%
25	150	10	81.21	104.20	0.11	79.96	104.20	4.42	1.56%	0.00%
50	245	5	37.67	44.40	0.24	21.85	44.40	3.84	72.36%	0.00%
50	367	5	28.84	39.00	0.21	17.30	39.00	6.28	66.74%	0.00%
50	490	5	23.87	37.40	0.33	15.21	37.40	8.26	56.94%	0.00%
50	612	5	24.25	29.60	0.55	12.27	29.60	7.90	97.61%	0.00%
50	245	10	78.73	96.60	0.28	62.62	96.60	9.10	25.73%	0.00%
50	367	10	73.89	80.40	0.33	51.44	80.40	9.93	43.65%	0.00%
50	490	10	66.12	81.00	0.80	44.98	81.00	36.40	47.00%	0.00%
50	612	10	54.61	69.80	0.90	36.82	69.80	25.07	48.30%	0.00%
50	245	15	152.49	176.20	0.34	125.39	176.20	10.38	21.61%	0.00%
50	367	15	129.13	136.60	0.41	102.33	136.60	14.19	26.19%	0.00%
50	490	15	112.05	134.20	0.74	88.59	134.20	31.59	26.49%	0.00%
50	612	15	100.14	111.60	0.96	76.12	111.60	28.93	31.55%	0.00%
50	245	20	223.05	228.60	0.28	188.52	228.60	6.47	18.32%	0.00%
50	367	20	193.58	205.00	0.48	165.63	205.00	14.56	16.88%	0.00%
50	490	20	148.33	161.40	0.65	125.47	161.40	22.71	18.22%	0.00%
50	612	20	157.06	171.60	0.80	133.60	171.60	22.24	17.56%	0.00%
75	555	8	48.51	58.60	1.05	23.79	58.60	20.18	103.93%	0.00%
75	832	8	41.99	62.80	2.55	27.91	62.80	82.42	50.46%	0.00%
75	1110	8	30.16	46.40	4.06	18.48	46.40	80.05	63.20%	0.00%
75	1387	8	34.45	43.60	2.71	20.04	43.60	44.49	71.87%	0.00%
75	555	15	97.19	121.40	0.95	79.93	121.40	40.24	21.59%	0.00%
75	832	15	94.64	111.80	5.12	70.67	111.80	145.38	33.91%	0.00%
75	1110	15	75.00	96.40	4.49	55.10	96.40	407.45	36.10%	0.00%
75	1387	15	60.35	90.80	7.68	47.56	90.80	701.17	26.88%	0.00%
75	555	23	187.52	222.40	3.53	162.47	222.40	154.83	15.42%	0.00%
75	832	23	158.94	178.00	2.94	127.54	178.00	402.89	24.62%	0.00%
75	1110	23	138.30	161.20	7.90	107.63	162.00*	973.69	28.49%	-0.49%
75	1387	23	131.89	147.00	3.70	105.30	147.00	254.37	25.25%	0.00%
75	555	30	291.01	307.20	1.70	237.69	307.20	109.54	22.43%	0.00%
75	832	30	204.28	234.00	1.45	181.19	234.00	68.79	12.74%	0.00%
75	1110	30	200.79	217.00	3.41	165.65	217.00	369.20	21.21%	0.00%
75	1387	30	186.20	208.40	8.96	153.63	209.00*	982.64	21.20%	-0.29%
100	990	10	51.71	70.80	4.94	26.39	70.80	258.00	95.95%	0.00%
100	1485	10	40.64	58.20	6.77	23.84	58.20	155.52	70.48%	0.00%
100	1980	10	29.90	52.00	10.54	22.68	52.00	381.85	31.85%	0.00%
100	2475	10	32.17	49.20	25.50	19.71	49.60*	1341.44	63.22%	-0.81%
100	990	20	128.05	149.60	5.48	91.30	149.60	343.70	40.26%	0.00%
100	1485	20	92.75	119.60	11.85	71.46	119.60	905.75	29.80%	0.00%
100	1980	20	76.00	121.40	22.04	64.89	122.40*	2300.55	17.12%	-0.82%
100	2475	20	64.00	103.60	23.79	56.79	108.60*	1947.42	12.70%	-4.60%
100	990	30	225.80	246.80	10.27	172.88	246.80	354.60	30.61%	0.00%
100	1485	30	184.57	211.80	8.67	149.75	211.80	440.12	23.25%	0.00%
100	1980	30	153.70	185.00	202.05	121.44	192.00*	2441.90	26.57%	-3.65%
100	2475	30	134.31	173.20	195.68	104.38	183.00*	3258.62	28.67%	-5.36%
100	990	40	315.86	341.00	3.77	270.11	341.00	232.40	16.94%	0.00%
100	1485	40	269.10	297.80	10.63	229.42	297.80	775.97	17.30%	0.00%
100	1980	40	215.75	237.60	7.72	179.59	237.60	325.57	20.14%	0.00%
100	2475	40	204.92	236.40	57.14	164.52	239.00*	3063.54	24.55%	-1.09%
150	2235	15	58.97	89.20	58.22	36.10	97.20*	3612.09	63.33%	-8.23%
150	3352	15	40.97	74.00	275.76	29.76	81.40*	3191.23	37.64%	-9.09%
150	4470	15	42.87	69.40	243.49	30.57	80.00*	3612.18	40.26%	-13.25%
150	5587	15	30.32	65.80	539.53	28.44	78.60*	3612.21	6.62%	-16.28%
150	2235	30	159.26	186.40	144.21	110.01	196.00*	3377.19	44.77%	-4.90%
150	3352	30	122.61	156.80	373.97	93.27	174.80*	3612.07	31.46%	-10.30%
150	4470	30	91.96	144.40*	1209.91	75.75	181.80*	3612.11	21.40%	-20.57%
150	5587	30	91.77	136.80	743.04	76.95	191.00*	3612.17	19.25%	-28.38%
150	2235	45	276.83	318.00	107.23	222.08	343.00*	3612.15	24.65%	-7.29%
150	3352	45	209.45	253.60	233.50	165.19	293.60*	3612.14	26.80%	-13.62%
150	4470	45	192.62	230.00	396.87	147.59	256.20*	3612.12	30.51%	-10.23%
150	5587	45	169.15	208.00	888.98	133.94	276.60*	3612.16	26.29%	-24.80%
150	2235	60	405.33	433.40	56.06	339.73	434.40*	2646.15	19.31%	-0.23%
150	3352	60	317.41	352.60	150.30	271.63	364.00*	2953.43	16.85%	-3.13%
150	4470	60	266.24	312.20	608.61	232.13	331.00*	3513.10	14.69%	-5.68%
150	5587	60	243.86	294.60	1066.29	198.13	454.25*	3612.22	23.08%	-35.15%

Table 5.2: Comparison of the ILP2-SV and ILP formulations for ACSP-SV on the new set of instances.

by ILP2-SV, the model requires up to about 18 minutes, while as mentioned the time limit is always reached by ILP.

A final comparison for the ACSP-SV problem is carried out between our VNS metaheuristic and those proposed in [3] for the problem. We recall that in this work the authors present 3 metaheuristics, namely a simulated annealing (SA), an ant colony optimization (ACO) and a genetic algorithm (GA). Moreover, they describe 3 heuristics based on iterative rounding of a mathematical formulation for the problem that they develop. The heuristics differ with respect to the variables on which the rounding is performed, and are called LP_x, LP_f and $LP_{f/x}$ respectively. These 6 algorithms were tested on the same dataset of Table 5.1. The results for these algorithms are taken from [3].

Table 5.3 contains the results of this comparison, with the results of the 4 metaheuristics (that is, our VNS and the ones in [3]) reported in the (a) subtable, and the rounding heuristics in the (b) subtable. In [3], for each instance and algorithm, the obtained result is reported in terms of proportion with respect to the optimal objective function value, found using CPLEX. Furthermore, since the 3 metaheuristics are non deterministic, the authors perform 10 independent runs for each of them and report the best and average solutions found, respectively. In order to be comparable, we ran our tests and reported our results accordingly; these values are contained in the *Avg Obj* and *Best Obj* columns for each metaheuristic, while the average computational times in seconds can be found in the *Avg Time* columns. The results for a single run was instead reported for each instance and each of the 3 heuristics, and we report these results in Table 5.3(b). In order to improve the computational times comparability (in [3], an AMD Phenom II X4 810 machine running at 2.67 GHz with 2 GB of RAM was used), we referred to the CPU performance comparative table provided by the UC Berkeley SETI@home experiment website[1], based on Whetstone benchmarks. By comparing the GFLOPS/core values, we divided all computational times reported in [3] by 1.21.

We can see that our VNS appears to be remarkably more effective than the previous approaches. The optimal solution is found in all 10 runs in one case ($n = 50$, $k = 10$), and in the best case for 5 out 8 instances. In the computational tests performed in [3], only SA (the most time-intensive approach) was able to find the optimal solution for the same instance in all 10 runs. The ACO algorithm was able to find the optimal solution

[1] https://setiathome.berkeley.edu/cpu_list.php

for this instance in the best case; no other instance was ever solved to optimality by any of their 6 proposed approaches.

On average, VNS found solutions diverging from the optimal one within 1% in 4 cases, 2% and 3% in one case each, and within 5% for the two largest instances with 200 vertices. In the best case, this threshold is never larger than 4%. On the other hand, the gap grow up to 46%, 55% or 58% for the previous 3 metaheuristics in the best case, and up to 50%, 62% or 72% in the average case. The overall best-performing rounding heuristic (LP_f) found a solution with an objective function gap equal to 15% for the instance with $n = 50$, $k = 10$, growing up to 39% for $n = 200$, $k = 50$.

With respect to the computational times, we note that GA algorithm appears to be the fastest one, running within 5 seconds on average. The ACO algorithm, while being slower than VNS on the smaller instances, appears to have roughly similar computational times on the largest ones; for $n = 200$, $k = 75$ the average computational time is 154.88 seconds for VNS and 139.81 seconds for ACO. The SA heuristic appears to be the most time intensive, being often at least one order of magnitude slower than VNS. The rounding heuristics have low computational times, being generally slower than GA and faster than ACO and VNS.

5.5.2 Comparisons on the ACSP problem

In this section, we use our VNS algorithm to solve the ACSP problem on our new instances, presented in the previous section. We compare the solution values found by VNS with the optimal values (or upper bounds) found by ILP2. In order to better verify the stability of the VNS and be consistent with the previously presented tests, we performed 10 independent runs of our metaheuristic on each instance. Table 5.4 presents the collected results. The first 3 columns contain the instance characteristics; the following 4 columns contain the results for our formulation (*ILP2* heading) and metaheuristic (*VNS* heading). Finally, the last column reports the gap value, in percentage, between the solutions of ILP2 and of VNS; in more detail, this value is computed as $100 \times \frac{Obj(VNS) - Obj(ILP2)}{Obj(ILP2)}$.

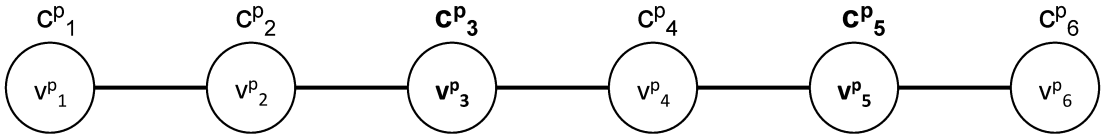
For each row, we present average values (in terms of objective function value and computational times in second) of all the computational tests performed for each scenario and for each of the two approaches.

The results reported in the *GAP* column show the effectiveness of VNS, that often finds either optimal solutions or very close ones. In particular, given the 74 scenarios that are solved by optimality by ILP2, we can see that the same solution is also found by VNS 23 times. Moreover, the gap value is lower than 1% for 53 out of 76 scenarios, and lower than 2% for 71 out of 76 scenarios. The gap value is higher than 3% only twice.

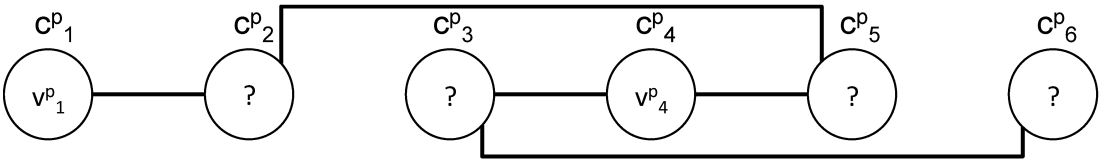
It is worth noting that on the smallest scenarios, with up to 75 nodes, the gap value is lower than 1% on 44 out of 46 scenarios; a single scenario ($n = 50$, $m = 612$, $k = 5$) among them has a peak corresponding to 3.54%, however we can note that the related solution values are small and therefore, in absolute terms, the solution values are not very far also in this case (22.60 for ILP2, 23.40 for VNS). Overall, the instance characteristics do not appear to influence the VNS performances in this case.

On the largest scenarios, gap value peaks occur instead on sparse scenarios with a number of colors equal to $\lceil 0.3n \rceil$ or $\lceil 0.4n \rceil$. In particular, the highest gaps can be noted for the following scenarios: $n = 100$, $m = 990$, $k = 30$ (1.65%); $n = 100$, $m = 990$, $k = 40$ (1.55%); $n = 150$, $m = 2235$, $k = 45$ (3.20%); $n = 150$, $m = 2235$, $k = 60$ (2.78%). The easiest scenarios for VNS are generally the ones containing less colors, where the gap values are almost always lower than 1%.

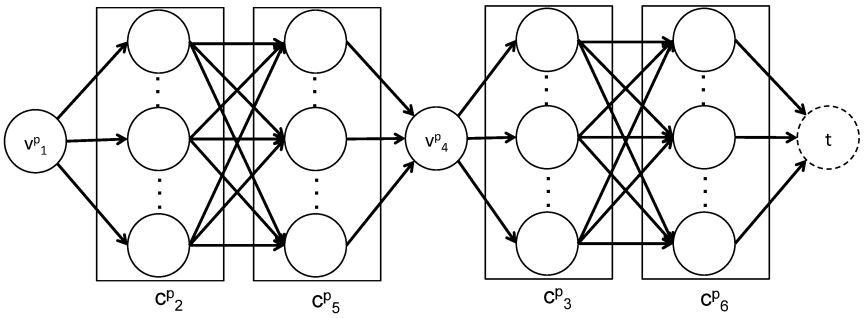
Regarding the performances, we can see that VNS runs in less than 5 seconds for the scenarios with up to 100 nodes, while for the largest ones it runs within 44 seconds. The parameter that mainly affects the performance is the number of colors. This was expected, since a higher number of colors leads to longer high-level solutions and therefore to a higher number of relocate and 2-opt operations. We can see that, for instance, all scenarios with $k = \lceil 0.1n \rceil$ are solved within 1 second, regardless of the number of nodes. The instances with a number of colors equal to $\lceil 0.2n \rceil$, $\lceil 0.3n \rceil$ and $\lceil 0.4n \rceil$ are instead solved within 5, 16 and 44 seconds, respectively.



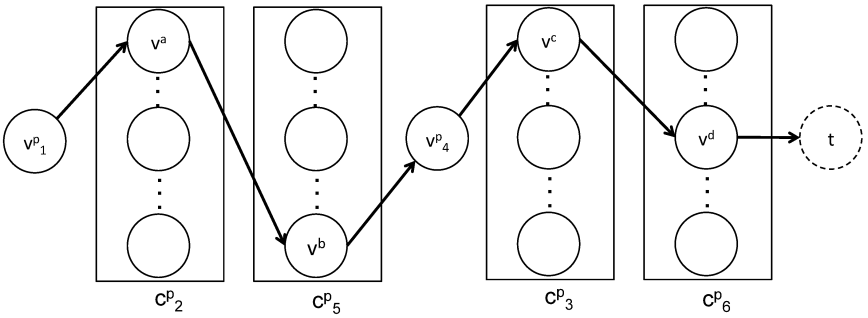
(a) Current high-level solution



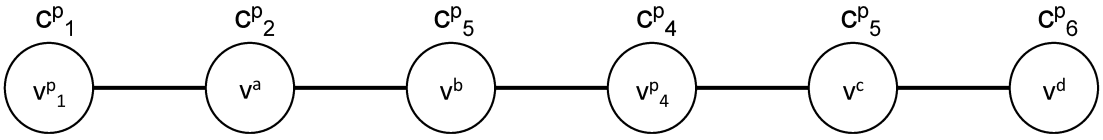
(b) Color sequence after 2-opt operation; new choices needed for c_2^p,c_5^p,c_3^p,c_6^p



(c) Auxiliary graph G^a (t=dummy destination)



(d) Shortest path identification



(e) High-level solution after the new vertex choices

Figure 5.5: Neighbor construction through 2-opt Local Search ($i = 3$, $j = 5$)

(a) Metaheuristics results

Instances			VNS			SA [3]		
n	m	k	Avg Obj	Best Obj	Avg Time	Avg Obj	Best Obj	Avg Time*
50	189	10	1.00	1.00	0.03	1.00	1.00	27.62
50	152	20	1.01	1.00	0.13	1.14	1.11	38.85
50	154	25	1.01	1.01	0.21	1.17	1.12	45.99
100	338	25	1.01	1.00	1.09	1.23	1.18	168.00
100	330	40	1.03	1.00	4.13	1.33	1.22	215.50
100	373	50	1.02	1.00	8.95	1.39	1.32	276.76
200	734	50	1.05	1.03	32.30	1.51	1.46	870.14
200	746	75	1.05	1.04	154.88	1.62	1.55	1231.93

Instances			ACO [3]			GA [3]		
n	m	k	Avg Obj	Best Obj	Avg Time*	Avg Obj	Best Obj	Avg Time*
50	189	10	1.06	1.00	1.92	1.07	1.05	0.35
50	152	20	1.19	1.17	6.74	1.23	1.12	0.45
50	154	25	1.18	1.12	11.46	1.33	1.22	0.59
100	338	25	1.26	1.17	11.63	1.22	1.15	0.79
100	330	40	1.33	1.28	31.41	1.54	1.40	1.08
100	373	50	1.40	1.29	57.74	1.56	1.45	1.90
200	734	50	1.44	1.39	52.43	1.58	1.45	2.48
200	746	75	1.50	1.46	139.81	1.72	1.58	4.71

(b) Rounding heuristics results

Instances			LP_x [3]		LP_f [3]		$LP_{f/x}$ [3]	
n	m	k	Obj	Time*	Obj	Time*	Obj	Time*
50	189	10	3.15	1.46	1.15	0.85	2.08	0.80
50	152	20	1.86	2.10	1.16	1.12	1.19	0.99
50	154	25	1.90	2.28	1.32	1.48	1.36	0.98
100	338	25	2.14	4.80	1.21	2.59	1.51	2.79
100	330	40	2.00	7.43	1.32	3.46	1.46	3.35
100	373	50	1.81	7.18	1.16	3.32	1.12	2.91
200	734	50	2.35	33.78	1.39	19.38	1.69	19.47
200	746	75	1.96	42.78	1.33	25.39	1.48	19.15

Table 5.3: Comparison of heuristics for ACSP-SV. Computational times reported in [3] are divided by 1.21

Instances			ILP2		VNS		GAP
n	m	k	Obj	Time	Obj	Time	
25	60	5	51.40	0.05	51.40	0.00	0.00%
25	90	5	37.60	0.09	37.60	0.00	0.00%
25	120	5	38.40	0.13	38.40	0.00	0.00%
25	150	5	33.20	0.11	33.20	0.00	0.00%
25	60	8	81.00	0.05	81.00	0.00	0.00%
25	90	8	88.40	0.13	88.40	0.00	0.00%
25	120	8	80.00	0.16	80.00	0.00	0.00%
25	150	8	79.20	0.19	79.40	0.00	0.25%
25	60	10	141.80	0.05	141.80	0.00	0.00%
25	90	10	123.20	0.13	123.60	0.00	0.32%
25	120	10	103.20	0.09	103.20	0.01	0.00%
25	150	10	87.80	0.12	87.80	0.00	0.00%
50	245	5	30.60	0.35	30.80	0.00	0.65%
50	367	5	24.80	0.40	24.80	0.01	0.00%
50	490	5	25.80	0.61	25.80	0.00	0.00%
50	612	5	22.60	1.27	23.40	0.00	3.54%
50	245	10	85.20	0.42	85.20	0.03	0.00%
50	367	10	70.20	0.52	70.20	0.03	0.00%
50	490	10	69.20	1.30	69.26	0.03	0.09%
50	612	10	63.80	1.78	63.80	0.03	0.00%
50	245	15	153.00	0.51	153.60	0.08	0.39%
50	367	15	125.80	0.89	126.24	0.09	0.35%
50	490	15	125.40	1.31	125.40	0.08	0.00%
50	612	15	104.80	1.81	104.80	0.07	0.00%
50	245	20	211.80	0.42	212.72	0.14	0.43%
50	367	20	191.20	0.54	194.92	0.14	1.95%
50	490	20	154.20	0.76	155.32	0.14	0.73%
50	612	20	159.60	0.94	160.30	0.13	0.44%
75	555	8	51.40	1.93	51.40	0.05	0.00%
75	832	8	46.00	1.83	46.28	0.05	0.61%
75	1110	8	36.80	3.52	36.80	0.05	0.00%
75	1387	8	36.80	5.20	36.80	0.05	0.00%
75	555	15	111.40	1.95	111.52	0.18	0.11%
75	832	15	103.20	6.08	103.20	0.19	0.00%
75	1110	15	89.80	8.46	90.50	0.18	0.78%
75	1387	15	83.00	10.27	83.00	0.17	0.00%
75	555	23	203.00	2.46	206.18	0.51	1.57%
75	832	23	173.20	4.71	174.58	0.45	0.80%
75	1110	23	153.40	11.71	153.90	0.43	0.33%
75	1387	23	139.60	6.51	140.04	0.47	0.32%
75	555	30	289.80	1.81	292.90	1.12	1.07%
75	832	30	228.00	2.57	229.92	0.99	0.84%
75	1110	30	208.00	4.07	209.90	0.97	0.91%
75	1387	30	198.00	7.53	199.40	0.92	0.71%
100	990	10	59.40	5.28	59.46	0.16	0.10%
100	1485	10	46.80	9.80	46.92	0.16	0.26%
100	1980	10	42.40	16.84	42.52	0.16	0.28%
100	2475	10	42.40	34.81	42.40	0.15	0.00%
100	990	20	138.40	11.63	140.10	0.64	1.23%
100	1485	20	111.00	34.22	111.54	0.60	0.49%
100	1980	20	110.60	48.98	111.40	0.60	0.72%
100	2475	20	96.80	40.00	98.02	0.57	1.26%
100	990	30	236.80	7.79	240.70	1.96	1.65%
100	1485	30	202.80	11.95	205.08	1.78	1.12%
100	1980	30	174.40	48.78	176.28	1.71	1.08%
100	2475	30	163.80	109.93	165.00	1.59	0.73%
100	990	40	324.40	5.78	329.42	4.67	1.55%
100	1485	40	287.60	12.42	291.84	4.37	1.47%
100	1980	40	229.80	17.77	231.90	3.82	0.91%
100	2475	40	228.40	40.50	230.94	3.89	1.11%
150	2235	15	77.00	47.50	77.46	1.00	0.60%
150	3352	15	66.60	150.67	67.12	0.97	0.78%
150	4470	15	60.20	300.98	60.26	0.93	0.10%
150	5587	15	56.80	383.15	57.46	0.92	1.16%
150	2235	30	180.00	182.17	181.96	4.52	1.09%
150	3352	30	145.60	210.39	146.60	4.27	0.69%
150	4470	30	134.80*	1042.86	136.82	4.15	1.50%
150	5587	30	130.00	1322.56	131.56	3.92	1.20%
150	2235	45	307.20	91.18	317.04	15.83	3.20%
150	3352	45	248.40	413.83	253.56	14.24	2.08%
150	4470	45	222.00	501.15	224.58	12.50	1.16%
150	5587	45	200.80	750.11	203.32	13.15	1.25%
150	2235	60	424.60	78.92	436.40	43.06	2.78%
150	3352	60	344.40	191.68	352.32	38.00	2.30%
150	4470	60	303.40	771.11	309.20	37.39	1.91%
150	5587	60	289.60*	1742.00	294.46	34.80	1.68%

Table 5.4: Computational results of VNS and ILP2 for the ACSP problem on the new set of instances.

Chapter 6

Conclusions

This last chapter contains a summary of the approaches developed to solve the presented subgraph identification problems. In particular, the conclusions are presented with respect to each chapter and some future research.

6.1 The Minimum Spanning Tree problem with Conflicting Edge Pairs (Chapter 3)

In this chapter, we studied the Minimum Spanning Tree problem with Conflicting Edge Pairs. Furthermore, we defined a variant of this problem named Minimum Conflict Weighted Spanning Tree problem, and we developed a genetic algorithm to solve it and three local search procedures to improve the quality of the solution found. To obtain a better exploration of the solution space, we embedded the genetic algorithm in a multi ethnic genetic framework.

The computational results show that Mega often finds the optimal solution or a solution close to the optimal one. Moreover, it found two new conflict free solutions with respect to the best known solutions in the literature. Finally, Mega significantly outperforms the tabu search heuristic, proposed in the literature, both in terms of computational time and quality of the solutions found.

To solve the Minimum Spanning Tree problem with Conflicting Edge Pairs problem, in the same chapter we described a novel Branch-and-Cut approach. In particular,

our main contribution is related to the proposal of a new set of valid inequalities, based on combined properties belonging to any feasible solution. Furthermore, we tested the approach we designed on the benchmark instances and compared it with a previous one. Our tests showed our approach to perform better on all instances except one, despite not using a preprocessing algorithm presented in the previous work in order to simplify the instances. Moreover, we created a new set of feasible instances, in order to test farther our approach and allow other researchers to have access to a wider set of benchmark instances for the problem. Future research will focus on finding new effective valid inequalities in order to improve our Branch-and-Cut approach.

6.2 The Close-Enough Arc Routing problem (Chapter 4)

We proposed a new MIP model for the CEARP based on a flow formulation and introduced some properties useful to reduce the size of the graph instances. For the benchmark instances, our graph reduction allowed to decrease the number of targets. This decrease ranges from 20% to 90% of the total number of them. The computational results show the effectiveness of our approach. For several instances, our approach is substantially faster than competing solution techniques.

6.3 The All Color Shortest Path problem (Chapter 5)

In this chapter we presented some new properties, as well as a mathematical formulation and a VNS metaheuristic for the all-colors shortest path problem. Our metaheuristic takes advantage of the concept of high-level solution, a fixed-length representation of any feasible solution. Our computational results shows that our approach outperforms significantly some previously introduced ones in the case in which the first vertex of the path is fixed, and that is able to find accurate solutions in fast computational times when the first endpoint is unconstrained. With respect to future developments, we intend to further study the problem and develop efficient exact approaches,

possibly based on Branch-and-Cut strategy. The development of new, more effective metaheuristics could also represent an interesting research direction.

References

[1] Whetstone benchmarks. 51

[2] Thais Ávila, Angel Corberán, Isaac Plana, and José M. Sanchis. A new branch-and-cut algorithm for the generalized directed rural postman problem. *Transportation Science*, 50(2):750–761, 2016. 73, 78

[3] Y. Can Bilge, D. Çagatay, B. Genç, M. Sari, H. Akcan, and C. Evrendilek. All colors shortest path problem. arXiv:1507.06865. 30, 80, 81, 82, 83, 98, 99, 102, 106

[4] B. Bontoux, C. Artigues, and D. Feillet. A memetic algorithm with a large neighborhood crossover operator for the generalized traveling salesman problem. *Computers and Operations Research*, 37(11):1844–1852, 2010. 31

[5] F. Carrabs, C. Cerrone, and R. Cerulli. A tabu search approach for the circle packing problem. In *2014 17th International Conference on Network-Based Information Systems*, pages 165–171. IEEE, 2014. 75

[6] F. Carrabs, R. Cerulli, P. Festa, and F. Laureana. *On the Forward Shortest Path Tour Problem*, volume Optimization and Decision Science: Methodologies and Applications: ODS, Sorrento, Italy, September 4-7, 2017, pages 529–537. Springer International Publishing, Cham, 2017. 30

[7] 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(2):405–438, 2013. 26

[8] Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Manlio Gaudioso. A novel discretization scheme for the close enough traveling salesman problem. *Computers & Operations Research*, 78:163–171, 2017. 29

[9] Francesco Carrabs, Carmine Cerrone, and Rosa Pentangelo. A multi-ethnic genetic approach for the minimum conflict weighted spanning tree problem. *Networks (under revision)*, 2017. 3

[10] Francesco Carrabs, Raffaele Cerulli, Rosa Pentangelo, and Andrea Raiconi. Properties, formulation and a two-level metaheuristic for the all-colors shortest path problem. *Computational Optimization and Applications (under revision)*, 2017. 3

[11] Francesco Carrabs, Raffaele Cerulli, Rosa Pentangelo, and Andrea Raiconi. Minimum spanning tree with conflicting edge pairs: a branch-and-cut approach. *Annals of Operations Research (in press)*, 2018. 3

[12] C. Cerrone, R. Cerulli, and M. Gaudioso. Omega one multi ethnic genetic approach. *Optimization Letters*, 10(2):309–324, 2016. 42

[13] Carmine Cerrone, Raffaele Cerulli, and Manlio Gaudioso. Omega one multi ethnic genetic approach. *Optimization Letters*, 10(2):309–324, 2016. 75

[14] Carmine Cerrone, Raffaele Cerulli, and Bruce Golden. Carousel greedy: a generalized greedy algorithm with applications in optimization. *Computers & Operations Research*, 85:97–112, 2017. 75

[15] Carmine Cerrone, Raffaele Cerulli, Bruce Golden, and Rosa Pentangelo. A flow formulation for the close-enough arc routing problem. In Antonio Sforza and Claudio Sterle, editors, *Optimization and Decision Science: Methodologies and Applications*, pages 539–546, Cham, 2017. Springer International Publishing. 3

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. 30, 92

[17] A. Darmann, U. Pferschy, and J. Schauer. Determining a minimum spanning tree with disjunctive constraints. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5783 LNAI:414–423, 2009. 25, 26, 32

[18] A. Darmann, U. Pferschy, J. Schauer, and G.J. Woeginger. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735, 2011. 25, 26, 32

[19] V. Dimitrijević and Z. Šarić. An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Information Sciences*, 102(1-4):105–110, 1997. 31

[20] Michael Drexl. On the generalized directed rural postman problem. *Journal of the Operational Research Society*, 65(8):1143–1154, 2014. 28, 73

[21] Michael Drexl and Hans-Jürgen Sebastian. On some generalized routing problems. Technical report, Deutsche Post Lehrstuhl für Optimierung von Distributionsnetzwerken (NN), 2007. 28, 73

[22] M. Dror, M. Haouari, and J. Chaouachi. Generalized spanning trees. *European Journal of Operational Research*, 120(3):583 – 592, 2000. 31

[23] J. Edmonds. Submodular functions, matroids and certain polyhedra. *Combinatorial structures and their applications*, pages 69–87, 1970. 24

[24] H.A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part i: The chinese postman problem. *Operations Research*, 43(2):231–242, 1995. 28

[25] H.A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part ii: The rural postman problem. *Operations Research*, 43(3):399–414, 1995. 28

[26] L. Euler. Solutio problematis ad geometrian situs pertinentis. *Commentarii academiae scientarum Petropolitanae*, 8:128–140, 1736. 27

[27] C. Feremans, M. Labbé, and G. Laporte. A comparative analysis of several formulations for the generalized minimum spanning tree problem. *Networks*, 39(1):29–34, 2002. 31

[28] C. Feremans, M. Labbé, and G. Laporte. The generalized minimum spanning tree problem: Polyhedral analysis and branch-and-cut algorithm. *Networks*, 43(2):71–86, 2004. 31

[29] P. Festa, F. Guerriero, D. Laganà, and R. Musmanno. Solving the shortest path tour problem. *European Journal of Operational Research*, 230(3):464–474, 2013. 30

[30] M. Fischetti, J. J. Salazar González, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995. 31

[31] M. Fischetti, J. J. Salazar González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997. 31

[32] Luisa Gargano, Pavol Hell, Ladislav Stacho, and Ugo Vaccaro. *Spanning Trees with Bounded Number of Branch Vertices*, pages 355–365. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. 25, 26

[33] A. M. H. Gerards and A. Schrijver. Matrices with the edmonds–johnson property. *Combinatorica*, 6(4):365–379, 1986. 49

[34] B. Golden, S. Raghavan, and D. Stanojević. Heuristic search for the generalized minimum spanning tree problem. *INFORMS Journal on Computing*, 17(3):290–304, 2005. 31

[35] M. Grotschel, L. Lovasz, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1987. 17

[36] M. Guan. Graphic programming using odd and even points. *Chinese Math.*, 1:273–277, 1962. 27

[37] Minh Hoang Há, Nathalie Bostel, André Langevin, and Louis-Martin Rousseau. An Exact Algorithm for the Close Enough Traveling Salesman Problem with Arc Covering Constraints. In *1st International Conference on Operations Research and Enterprise Systems (ICORES)*, pages 233–238, Portugal, February 2012. 73, 78

[38] Minh Hoang Há, Nathalie Bostel, André Langevin, and Louis-Martin Rousseau. Solving the close-enough arc routing problem. *Networks*, 63(1):107–118, January 2014. 73, 78

[39] M. Haouari, J. Chaouachi, and M. Dror. Solving the generalized minimum spanning tree problem by a branch-and-bound algorithm. *Journal of the Operational Research Society*, 56(4):382–389, 2005. 31

[40] J. H. Holland. Adaption in natural and artificial systems. *Ann Arbor MI: The University of Michigan Press*, 222(3):2–5, 1975. 35

[41] B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499, 2008. 31

[42] E. Ihler, G. Reich, and P. Widmayer. Class steiner trees and vlsi-design. *Discrete Applied Mathematics*, 90(1–3):173 – 194, 1999. 31

[43] M. M. Kanté, C. Laforest, and B. Momège. Trees in graphs with conflict edges or forbidden transitions. *Theory and Applications of Models of Computation: 10th International Conference, TAMC 2013, Hong Kong, China, May 20-22, 2013. Proceedings*, pages 343–354, 2013. 26

[44] A. Klein, D. Haugland, J. Bauer, and M. Mommer. An integer programming model for branching cable layouts in offshore wind farms. *Advances in Intelligent Systems and Computing*, 359:27–36, 2015. 27

[45] G. Laporte, H. Mercure, and Y. Nobert. Generalized travelling salesman problem through n sets of nodes: the asymmetrical case. *Discrete Applied Mathematics*, 18(2):185 – 197, 1987. 31

[46] B.D.H Latter. The island model of population differentiation: a general solution. *Genetics*, 73(1):147–157, 1973. 42

[47] J. K. Lenstra and A. H. G. Rinnooy Kan. On general routing problems. *Networks*, 6:273–280, 1976. 28

[48] Oliver Lum, Carmine Cerrone, Bruce Golden, and Edward Wasil. Partitioning a street network into compact, balanced, and visually appealing routes. *Networks*, 69(3):290–303, 2017. 73

[49] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995. 31

[50] Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995. 25, 26

[51] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial optimization*. Wiley, New York, NY, USA, 2014. 17

[52] T. Öncan, J.-F. Cordeau, and G. Laporte. A tabu search heuristic for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 191(2):306–319, 2008. 31

[53] T. Oncan, R. Zhang, and A.P. Punnen. The minimum cost perfect matching problem with conflict pair constraints. *Computers and Operations Research*, 40(4):920–930, 2013. 32

[54] M.W. Padberg and L.A. Wolsey. Trees and cuts. *North-Holland Mathematics Studies*, 75(C):511–517, 1983. 49, 50

[55] U. Pferschy and J. Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13(2):233–249, 2009. 32

[56] U. Pferschy and J. Schauer. The maximum flow problem with disjunctive constraints. *Journal of Combinatorial Optimization*, 26(1):109–119, 2013. 32

[57] P. C. Pop, W. Kern, and G. Still. A new relaxation method for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 170(3):900–908, 2006. 31

[58] R. Sadykov and F. Vanderbeck. Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255, 2013. 32

[59] P. Samer and S. Urrutia. A branch and cut algorithm for minimum spanning trees under conflict constraints. *Optimization Letters*, 9(1):41–55, 2014. 32, 44, 45, 49, 58

[60] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, New York, NY, USA, 1998. 17

[61] X.H. Shi, Y.C.Liang, H.P.Lee, C.Lu, and Q.X.Wang. Particle swarm optimization-based algorithms for TSP and generalized TSP. *Information Processing Letters*, 103(5):169–176, 2007. 31

[62] Robert Shuttleworth, Bruce L. Golden, Susan Smith, and Edward Wasil. *Advances in Meter Reading: Heuristic Solution of the Close Enough Traveling Salesman Problem over a Street Network*, pages 487–501. Springer US, Boston, MA, 2008. 73

[63] L. V. Snyder and M. S. Daskin. A random-key genetic algorithm for the generalized traveling salesman problem. *European Journal of Operational Research*, 174(1):38–53, 2006. 31

[64] D. Whitley, S. Rana, and R.B Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of computing and information technology*, 7(1):33–48, 1999. 42

[65] L.A. Wolsey. *Integer Programming*. Wiley, Hoboken, New Jersey, 1998. 17

[66] R. Zhang, S.N. Kabadi, and A.P. Punnen. The minimum spanning tree problem with conflict constraints and its variations. *Discrete Optimization*, 8(2):191–205, 2011. 27, 32, 35, 44, 51, 53