University of Salerno

Department of Mathematics

PhD in Mathematics, Physics and Applications

Curriculum: Mathematics

CYCLE: XXXV

PhD Thesis

Hidden Structures and Conflicting Items in Combinatorial Optimization Problems

Tutor
Prof. Raffaele Cerulli

Co-Tutor
Prof. Francesco Carrabs

Coordinator
Prof.ssa Patrizia Longobardi

Candidate
Domenico Serra

2021-2022

Alla mia Famiglia e Alessandro

Acknowledgement

I am extremely grateful to my mentor and Tutor, Prof. Raffaele Cerulli, for his guidance, patience, constant support, and valuable advice over the last three years. Thank you for everything.

I also thank my Co-Tutor, Prof. Francesco Carrabs, for his assistance at every stage of my academic life.

I am deeply grateful to Claudia Archetti for hosting me during my research period abroad. I also want to thank Prof. Ivana Ljubic for contributing to my research work abroad.

Many thanks to Martina Cerulli for the work done together and for the numerous pieces of advice she gave me.

Special thanks to my colleague Carmine Sorgente for the friendship and support shared during these years.

During my PhD, I met wonderful colleagues whom I sincerely thank: Federica Laureana, Ciriaco D'Ambrosio and Andrea Raiconi, thanks for the time spent together.

I also want to thank Prof. Renata Mansini and Lorenzo Moreschini of the University of Brescia for the work done together on the Set Covering problem with Conflicts on Sets.

Special thanks to new and old friends who have always supported me. Thanks to Roberta Guadagni, Federico Vitale and Carmine Sorgente, friends since my early university years.

I thank Lidia Ponzo for being a loyal friend and always supporting me. I will never forget all the time we spent together.

Special thanks to Graziano Fuccio, my greatest flatmate with whom I spent unforgettable moments.

Among the new friends, I want to thank Fabio Capone. He is a nice person, and we found ourselves immediately in tune.

I want to thank my wonderful family for their constant love and support. They always believe in me, accept my choices and make me feel safe. Special thanks go to my great sister Angela, she always cheers for me.

A final thank goes to Alessandro. I am so grateful to have you as my partner. Thank you for always being there for me, no matter what.

Abstract

Combinatorial optimization-based methods are widely used to solve complex real life problems. In this thesis, we use some of these methods for addressing several emerging combinatorial optimization problems on graphs that can be classified in two macro areas: *i)* graph substructures identification problems; *ii)* combinatorial optimization problems with conflict constraints.
In graph theory, graphs are defined as mathematical structures that describe entities of interest (as nodes) and their relationships (as edges). Many real-world problems can be described through the use of a graph. For example, graph theory finds application in: *i)* the context of social network analysis, where graphs are used to represent the interactions (edges) between users (nodes); and in *ii)* the study of biological networks, such as protein-protein interaction, where the nodes are proteins, and the edges represent their physical interactions. In graph analysis one common application is the identification of clusters (or communities) of nodes that are tightly connected. In social networks, a community could represent a set of users sharing the same interest, while in the protein-protein interaction networks it could represent a set of very similar proteins forming a protein complex.

In this thesis, three problems related to identifying graph substructures have been tackled.

The first problem addresses the 2-Edge-Connected Minimum Branch Vertices, that finds application in the design of optical networks. A graph is 2-Edge-Connected if by removing one edge, the graph is still connected. The problem looks for a spanning 2-edge connected subgraph having the minimum number of branch vertices that is vertices with degree strictly greater than two. In these networks, branch vertices are associated with switch devices that split the light signals and send them to the adjacent vertices. For this NP-complete problem we developed a genetic algorithm using ad-hoc designed operators.

The second addressed problem arise in the social network analysis and aims to study how users influence the choices of their neighbours. In particular, we addressed the Collapsed k-Core Problem that seeks to identify a subset of critical users in the network whose choices would alter the cohesiveness of a community. To the best of our knowledge, this is the first attempt to formulate this problem using mathematical programming. We implemented multiple solution approaches and compared them on a set of benchmark instances.

The last case studied is related to network clustering, where a cluster graph is a disjoint union of cliques. The Cluster Deletion problem is defined as the identification of the minimum number of edges to remove from a network to produce a cluster graph. This is a well-known NP-hard problem and we faced it using integer linear programming formulation and a heuristic approach based on edge contraction operation. Our results show the effectiveness of our methodology both on artificial and real-world biological networks.

Currently, the definition of many real-life problems, doesn't always fully capture their complexity. Indeed, classical optimization problems encountered over the years, although extensively studied, do not always take into account additional limitations, such as incompatibility situations, encountered in real-world problems. In this context, two or more elements of the problem cannot be chosen together to compose a feasible solution. Such incompatibilities are modelled by introducing conflict constraints in classical combinatorial optimization problems, leading to more realistic but often harder problems.

Finally, three different combinatorial optimization problems with conflicts constraints are addressed.

The first one is a variant of the set cover problem where pairwise conflicts are added among the subsets. In the formulation of this problem, two sets in conflict can belong to the same solution, provided that a non-negative penalty is paid. We introduced two mathematical formulations for the problem and offered a parallel Greedy Randomised Adaptive Search Procedure for its solution. The performance of our algorithm was evaluated through an extensive set of experiments. The results shown the effectiveness and efficiency of our methodology compared to the mathematical model solutions.

The second problem is related to the Maximum Flow Problem with Conflict constraints on the edges, for which we present a matheuristic method based on the combination of two different approaches: Carousel Greedy and Kernel Search. The results shown that the Carousel Greedy selection substantially improves the effectiveness of the Kernel Search.

The last problem is the Minimum Spanning Tree with Conflicts, that we solved by using a Kernel Search method. Also in this case, the results on benchmark instances shown that our methods identifies a better solution compared with existing methods.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are abstract models to represent connections between different entities and are successfully applied to solve real instances of optimization problems. A graph can be *i)* undirected, in this case no direction is assigned to each edge *ii)* directed, in this case a direction is assigned to each edge. An undireceted (directed) graph is defined by a pair of sets, i.e. $G(V, E)$ $(G(V, A))$, where V is the set of the nodes and E (A) is the set of the undirected (directed) edges.

Graphs have long been used to solve problems and challenges humanity has faced. One of the firsts formulation and definition of the Graph Theory can be traced back to Euler's resolution of the problem of the Königsberg bridges in the 18th century, in this problem the river Pregel and its tributaries cross the city of Königsberg. The city is distributed along the river on the mainland and on two large islands connected to each other and to the two remaining areas of the city by seven bridges. The problem that arose was whether it was possible to build a path that ended at the point where it started by crossing each bridge exactly once.

Another exciting application of graphs that laid the foundations of part of modern graph theory is the icosahedron game proposed by Hamilton in 1859. Hamilton assigned the name of a city to each vertex and tried to find a route that visited all the cities only once and then returned to the starting point. Hence the term Hamiltonian circuit or Hamiltonian cycle was coined as the precursor of the modern problem of the TSP (Travelling Salesman Problem), considered one of the hardest problem in combinatorial optimization. The TSP consists in finding the minimum length cycle that traverses each node exactly once in a complete weighted graph.

Mathematics has undergone enormous development in the decades that separated the last quarter of the 19th century from the Second World War, profoundly transforming its nature and image. New research areas were born, including *operations research*, triggered by the need to face logistical problems with limited resources.

An essential part of operations research is *optimization* or *mathematical programming*, whose spread accelerated with the increasing amount of computers available. *Optimization* covers thousands of applications in multiple fields such as chemistry and physics, computer networking, most branches of engineering, manufacturing, public policy, social systems, scheduling and routing,

telecommunications, and transportation. Many problems related to operations research, can be modelled using graphs. One of the first important problems studied through graph theory is the Maximum Flow problem. The Maximum Flow problem was first formulated in 1954 by T. E. Harris and F. S. Ross to solve the problem of simplify the flow model of the Soviet railway system [Sch02]. In 1955, Lester R. Ford, Jr. and Delbert R. Fulkerson published the first known algorithm to solve Max Flow problem (Ford-Fulkerson algorithm [FF56]). Since that time many graph problems have been solved, formulated as maximum flow problems (management of electricity networks, water networks, traffic networks, etc.).

Graphs certainly represent an efficient way to represent complex problems. Currently, the solution approaches of many real problems are based on their representation through graphs.

This thesis deals with new optimization problems defined on graphs; in particulare we will face problems in which we need to identify particular substructures in graphs. These problems include the design of wired optical networks, interdiction problems in social networks and clustering problems in biological networks.

Moreover, we will study combinatorial optimization problems in which there are conflict constraints. Conflict constraints are used to describe real-world incompatibility situations that sometimes make the classical solutions unfeasible.

In the following we briefly describe the two classes of graphs problems that we studied: *i)* Identification of substructures in graphs; *ii)* Combinatorial optimization problems with conflict constraints.

Identification of substructures in graphs

In this thesis we faced three important problems, related to the Identification of substructures in graphs: *i)* 2-Edge-Connected Minimum Branch Vertices Problem; *ii)* Collapsed k-core Problem; *iii)* Cluster Deletion Problem.

i) The Minimum Branch Vertices problem (MBV), introduced by Gargano et al. [GHSV02], has applications in optical network design. Given an undirected graph, the goal of the MBV problem is to find a spanning tree with the minimum number of *branch* vertices. A vertex is called *branch* if it has more than two incident edges. This type of vertex in an optical network represents an expensive connection. An interesting variant of the MBV problem has been dealt with in this thesis and consists of the 2-Edge-Connected Minimum Branch Vertices Problem (2ECMBV), proposed for the first time by Laureana [Lau19] in her PhD thesis, for which Laureana proposed a Branch and Cut algorithm able to solve small size instances. The 2ECMBV problem consists in the identification of a 2-edge-connected spanning sub-graph having the minimum number of branch vertices. A graph is said to be 2-edge-connected if, in any way we remove strictly less than two edges, the graph is still connected. To address this problem, we implement

a genetic algorithm (GA), that we test on a set of instances proposed by Laureana [Lau19] and a new set of instances randomly generated.

ii) In recent years the study of the behaviour of users in social networks has gained increasing interest. In this context particularly important are the so-called critical users, i.e., the ones who have many connections with other users and whose departure from the network might potentially cause the exit of many other users. Users who leave a community potentially affect the cardinality of its k-core, i.e., the maximal induced subgraph of the network with a minimum degree of at least k. In social network analysis, the size of the k-core is frequently adopted as a typical metric to evaluate the cohesiveness of a community. We focus on the Collapsed k-Core Problem, which seeks to find a subset of b users, namely the most critical users of the network, the removal of which results in the smallest possible k-core. We model the Collapsed k-Core Problem as a natural deletion-round-indexed Integer Linear programming formulation and we provide two bilevel programs for the problem, which differ in how they formulate the k-core identification problem. We reformulate the first bilevel formulation as a single-level sparse model, exploiting a Benders-like decomposition approach. To derive the second bilevel model, we first provide a linear formulation for finding the k-core of a network and use it to state the lower-level problem and then dualize the lower level and obtain a compact Mixed-Integer Nonlinear single-level problem reformulation. Moreover we derive a combinatorial lower bound on the value of the optimal solution and describe some pre-processing procedures and valid inequalities for all three formulations. The performance of the solution approaches of the two proposed formulations are compared on a set of instances using several network data sets available in the literature. Furthermore, we compare our approaches with the existing state-of-the-art solver for mixed-integer bilevel problems proposed in [FLMS17].

iii) Clustering can be used to identify substructures in graphs. We refer to a cluster graph as a disjoint union of cliques obtained by clustering the nodes of a given network and removing the edges that not belong to any clique. A clique is a collection of nodes in an undirected graph, such that every two different nodes in the clique are connected through an edge. We addressed the Cluster Deletion Problem (CD), which asks for the minimum number of edges to be removed from a network to produce a cluster graph (this problem is equivalent to determining the maximum number of edges to be preserved). Many fields could benefit from the CD problem: computational biology, bioinformatics, wireless sensor networks, etc. To solve this problem we provide two Integer Linear Programming formulations and we provide an heuristic approach.

Combinatorial optimization problems with conflict constraints

Conflict constraints, also named incompatibility constraints, are widely used in various combinatorial problems, such as knapsack problems [CFSS21, HM07, PS09], shortest path problems [DPSW11], matching problems [ÖKA18, OZP13], arc routing problems [CCM+17], bin packing problems [Eki21, EFL11, SV13], minimum spanning tree problems [CCP19, CCPR21b, CG21a]

and maximum flow problems [PS13a, cAA20].

In this thesis we faced three combinatorial optimization problems with conflict constraints: *i)* The Set Covering Problem with Conflicts on Sets; *ii)* The Maximum Flow Problem with Conflicts; *iii)* The Minimum Spanning Tree Problem with Conflicts.

i) We analyze a new variant of the well-known Set Cover Problem characterized by pairwise conflicts among subsets. Two conflicting sets can belong to the same solution, provided that a non-negative penalty is paid. To solve this problem we want to identifies the optimal collection of subsets representing a cover that minimizes the sum of covering and penalty costs. For this problem called the Set Covering problem with Conflicts on Sets (SCCS), we propose two mathematical formulations and we design a GRASP technique approach. Computational results show that the GRASP algorithm is highly effective compared whit the solution obtained using mathematical model.

ii) We address a variant of the Maximum Flow Problem, in which given pairs of arcs are not allowed to bring positive flow simultaneously. This problem is known to be strongly NP-hard, and some exact approaches has been proposed in the literature [cAA20]. To address this problem we design a heuristic and matheuristic approach: the heuristic approach is based on a variant of Greedy algorithm called Carousel Greedy algorithm; the matheuristic combines the Carousel Greedy and the Kernel Search approach together.

iii) The Minimum Spanning Tree Problem with Conflicts consists of finding the minimum conflict-free spanning tree of a graph, i.e., the spanning tree of minimum cost, whit no pairs of conflicting edges. For this problem we design a tailored Kernel Search.

The remaining chapters of this thesis are dedicated to the addressed problems, while the readers are reminded to Appendix A for a detailed description of the solution approaches, used to solve the optimization problems addressed in this thesis. In particular: in Chapter 2, we study the 2-Edge-Connected Minimum Branch Vertices Problem (2ECMBV); in Chapter 3 we study the Collapsed k-Core Problem; in Chapter 4 we study the Cluster Deletion Problem; in the Chapter 5 we study the Set Covering Problem with Conflicts on Sets; in Chapter 6 we study the Max Flow Problem with Conflicts; in Chapter 7 we study the Minimum Spanning Tree Problem with Conflicts and in Chapter 8, we report the conclusion of the thesis.

Part I

Identification of substructures in graphs

Chapter 2

2-Edge-Connected Minimum Branch Vertices Problem

In this chapter we deal with the first substructure identification problem, which is the 2-Edge-Connected Minimum Branch Vertices Problem (2ECMBV). A graph is said to be 2-edge-connected if, in any way we remove one edge, the graph is still connected. This problem has already been described in Chapter 1 and looks for a spanning 2-edge connected subgraph having minimum number of vertices with degree strictly greater than two (branch vertex). This problem finds application in the design of optical networks. In an optical network, the wave division multiplexing technology, enables the transmission of multiple light beams with various fixed wavelengths over the same optical fiber. Through the use of a network device called a *switch*, multicast technology on an optical network enables the replication of the optical signal from a single source to numerous destination vertices. This allows the network device to split an incoming light signal and send it to more adjacent vertices. To enable multicasting communications, a light-tree connects the network's vertices. The vertices of the tree with a degree greater than two are referred to as *branch* vertices and need a switch to split and propagate the light signal to the neighboring vertices, so this types of nodes are generally expensive. The number of switches should be kept to a minimum due to budgetary restrictions, so the goal of the problem is to identify the network's spanning tree with the fewest possible branch vertices.

The search for a 2-edge-connected structure is motivated by the will to obtain a structure that remains connected when a single connection loss occurs. Similar problems have already been addressed in the literature through the identification of a weaker structure, such as a tree, which is unable to cope with failure..

2ECMBV is a variant of the Minimum Branch Vertices (MBV) problem introduced for the first time by Gargano et al. [GHSV02]. MBV consists of finding a spanning tree, with the minimum number of branch vertices, which are vertices having a degree greater than two in the spanning tree.

Carrabs et al. [CCGG13], introduced four IP formulations for the MBV problem. Silvestri et al. [SLC17] proposed a Branch-and-cut approach for MBV and they introduced some valid

inequalities for this problem. Some heuristic approaches for the MBV have also been proposed in the literature, such as an edge-swap heuristic algorithm proposed by Silva et al. [SSR+14], Cerulli et al. [CGI09] while Marín [Mar15] experimented both exact and heuristic approaches. In this Chapter, we demonstrate that 2ECMBV is NP-complete through a polynomial reduction from the MBV problem. Furthermore, we developed an algorithm that finds and eliminates redundant edges from a feasible solution in order to reduce the number of its branch vertices by using some 2-edge connectivity properties. Finally we propose a genetic algorithm (GA) based on ad-hoc operators designed to widely explore the solution space. The computational results carried out on benchmark and new instances, show that our algorithm is fast and effective because, in the worst case, it returns solutions having one branch vertex more than the optimal/best ones.

2.1 Notation and Definitions

Let $G = (V, E)$ be an undirected, unweighted, connected graph, where V is the set of vertices, E is the set of edges, $|V| = n$ and $|E| = m$. Given a vertex $v \in V$, we denote by $d_G(v)$ the degree of v in G and by $\delta_G(v)$ the set of edges incident to v in G. Moreover, given a collection of vertices $W \subset V$, we indicate by $\delta_G(W)$ the set of edges of G that connect any vertex in W with a vertex in $V \setminus W$, and by $G[W]$ the subgraph of G induced by the vertices in W. A vertex $v \in V$ is a *branch vertex* if it has a degree more than two. Furthermore, G is *2-edge-connected* if and only if, the resulting subgraph is still connected, by removing one edge. The edge connectivity version of Menger's theorem [RTJ93] leads to a further characterization of the 2-edge-connectivity property: a graph results to be 2-edge-connected if at least two edge-disjoint paths between each pair of vertices exist. In the following, we only consider simple paths, i.e. paths without repeated vertices. The *2-Edge-Connected Minimum Branch Vertices* (2ECMBV) problem consists in finding a 2-edge-connected spanning subgraph G' of G having the smallest number of branch vertices.

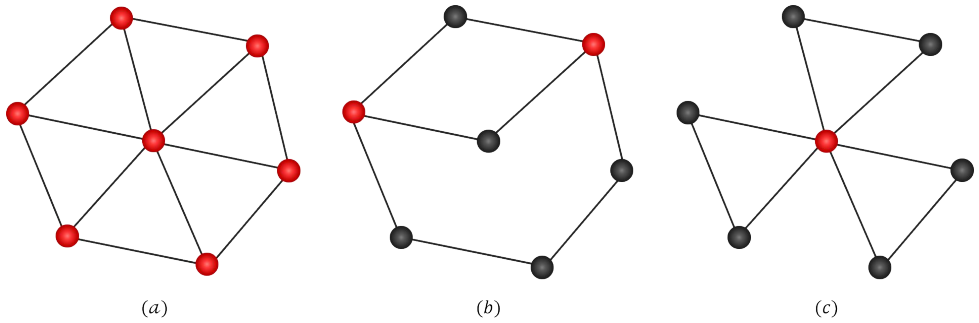


Figure 2.1: (a) A connected, undirected and unweighted graph G. The branch vertices are shown in red. (b) A 2-edge-connected spanning subgraph of G with two branch vertices. (c) A 2-edge-connected spanning subgraph of G with one branch vertex.

Given the graph G in Figure 2.1(a), there are two feasible solutions for the 2ECMBV problem presented in Figures 2.1(b) and 2.1(c). We can note that, the subgraph in Figure 2.1(c) represents a better solution compared to the one in Figure 2.1(b) because it contains one branch vertex less.

2.2 Complexity analysis

We prove that the 2ECMBV belongs to the class of problems NP-complete. To this end, we take into account the corresponding decision version of the problem and, after showing that it is in NP, we provide a polynomial reduction from the MBV problem, which is known to be NP-complete [GHSV02].

Here we report the decision version of the 2ECMBV and MBV problems:

MBV$_\mathbf{d}$: *Given a connected, undirected graph G, is there a spanning tree T of G, having at most b branch vertices?*

2ECMBV$_\mathbf{d}$: *Given a connected, undirected graph $G = (V, E)$, is there a 2-edge-connected subgraph $G' = (V, E')$, with at most b branch vertices?*

Is easy to verify the feasibility of a given solution of the 2ECMBV problem and it can be done in polynomial time. To certify the feasibility of a solution we have to: *i)* check the subgraph 2-edge-connectivity; *ii)* check the subgraph spanning property; and *iii)* count the number of branch vertices in the subgraph.
The step *i)* can be done in $O(m + n)$ using Tarjan algorithm [Tar74] to find bridge edges in a graph; The step *ii)* can be performed with a breadth-first search of the subgraph, while step *iii)* is linear in the number of vertices.

We transform in polynomial time an instance of MBV_d into an instance of $2ECMBV_d$ in order to demonstrate the hardness of the 2ECMBV problem. The idea is that, the answer of the $2ECMBV_d$ problem on this instance is YES if and only if the answer of the MBV_d problem is YES on the original instance.

Proposition 2.2.1. *Given an instance (G, b) of MBV_d, such that G contains at least two vertices, $MBV_d \leq_\mathbf{p} 2ECMBV_d$.*

Proof. Given an instance of MBV_d (G, b), we build an instance (G', b') of $2ECMBV_d$ as follows. We put in G' three copies of G: $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ and $G_3 = (V_3, E_3)$. In addition, three vertices are added to G' that are directly connected to each other: one for each copy of G. Each vertex v_i is connected, with additional edges, towards each vertex of its corresponding copy G_i. This procedure is illustrated in Figure 2.2. The value of b' is set to $3b + 3$.

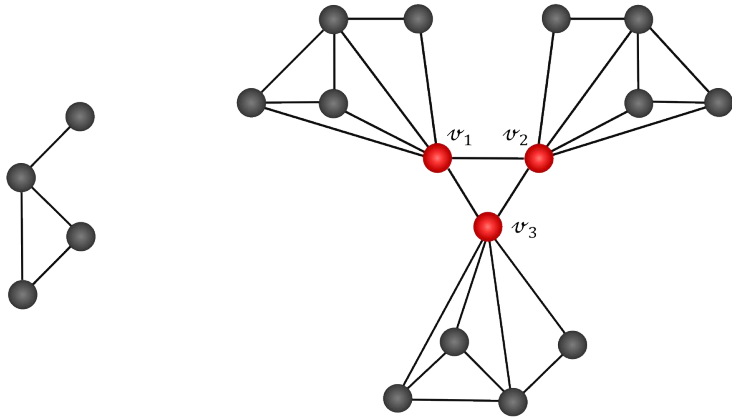


Figure 2.2: (a) Input graph G of the MBV_d problem (or instance grap). (b) Instance graph G' of the $2ECMBV_d$ problem built from G, in order to support the reduction $MBV_d \leq_p 2ECMBV_d$. The vertices v_1, v_2 and v_3 are used to connect three copies of G.

Let T a spanning tree for G supposing having at most b branch vertices and let T_1, T_2 and T_3 be the three copies of T respectively in G_1, G_2 and G_3. By construction, each T_i has at most b branch vertices. By linking each leaf of T_i with the corresponding vertex v_i, the subgraph induced by $T_i \cup \{v_i\}$ is 2-edge-connected with at most $b+1$ branch vertices because the selected edges from v_i to the leaves do not transform these last vertices in branch vertices. It is easy to see that the subgraph induced by $T_1 \cup T_2 \cup T_3 \cup \{v_1, v_2, v_3\}$ is a 2-edge-connected subgraph of G' and has $3b+3$ branch vertices, at most.

Let $T = (V_T, E_T)$ be a 2-edge-connected spanning subgraph of G' having at most b' branch vertices. It is easy to see that v_1, v_2 and v_3 are branch vertices in T. This means that the number of branch vertices in the subgraph of T induced by $\bigcup_{i=1}^{3} V_i$ is at most equal to $(b'-3)$. As a consequence, among the sets of vertices V_1, V_2 and V_3, there must be at least one of them, w.l.o.g let us suppose V_1, such that the subgraph of T induced by V_1 contains at most $(b'-3)/3$ branch vertices. Then, this subtree is a solution for MBV_d (G, b). □

2.3 2-Edge-Connectivity Operators

In this section we analyze two operators implemented in our GA in order to ensure the feasibility of the chromosomes in the genetic population. The first operator verifies the 2-edge-connectivity of a solution, while the latter restores this property, when needed.

The first operator is the ***2-Edge-Connectivity Checker Operator***. There are various algo-

rithms in the literature for determining if a graph is 2-edge-connected. Tarjan [Tar72] proposed an algorithm that run in $O(m+n)$ time, that find biconnected components of a graph. This approach may be used to detect whether a graph is 2-vertex-connected. Galil et al. [GI91] polynomially reduced 2-edge-connectivity to 2-vertex-connectivity, which means that any known procedure for verifying 2-vertex-connectivity, including the algorithm proposed by Tarjan, may also be used to check 2-edge-connectivity. Tarjan [Tar74] also developed a $O(m+n)$ technique for determining graph bridges. If the number of connected components in $(V, E \setminus e)$ is strictly higher than the number of connected components in G, an edge $e \in E$ is a bridge in $G = (V, E)$. This approach may also be used to assess 2-edge-connectivity since a connected graph with at least two vertices is 2-edge-connected if and only if it has no bridges. Furthermore, the Schmidt [Sch13] algorithm, which exploits chain decomposition to find bridges and cut vertices in a graph, can check 2-connectivity and 2-edge-connectivity in $O(m+n)$ time. More precisely, after performing a depth-first search, the fundamental cycles in the resulting depth-first search tree are considered, and each one if it does not overlap with any previously added cycle is added to the chain decomposition; otherwise, only the initial non-overlapping segment is taken into account. A bridge is an edge that is not contained in any chain of the decomposition; similarly, a connected graph with at least two vertices is 2-edge-connected if its chain decomposition splits its set of edges. We use the Schmidt approach to evaluate the 2-edge-connectivity of subgraphs in the construction of our genetic algorithm.

The second operator is the: ***2-Edge-Connectivity Restorer Operator***. We can restore the 2-edge-connectivity property for a subgraph of $G = (V, E)$, when it does not hold, by adding edges from $E \setminus E'$ to the E'. Eswaran et al. [ET76] demonstrated that when every edge in the complement graph of G' is available, the least number of edges to be added into E' to make the subgraph G' 2-edge-connected, can be found in $O(m+n)$. However, because we need to avoid selecting non-existing edges in the original graph G and we want to minimize the number of branch vertices in the new subgraph, we solve the weighted version of this problem, which is NP-hard and aims to identify a minimum-weight edge set that makes G' 2-edge-connected.
Despite the fact that graphs in the 2ECMBV problem are unweighted, we give every edge $e \in E$ a weight by doing the following: *(i)* $w_e = \infty$ if $e \notin E \setminus E'$; *(ii)* $w_e = 1 + \beta$ otherwise, where $\beta \in \{0, 1, 2\}$ is the number of no branch vertices incident on e. By doing this, we promote choosing edges that coincide with vertices that are already branches since they reduce the cost of a solution. We use an implementation of the 2-approximation technique suggested by Khuller et al. [KT93], which accomplishes the task of selecting high-quality edge augmentations for G' in a short time ($O(m + n \log n)$). In the remaining of the thesis, we refer to the invocation of such procedure as the restorer operator, and denote it by `Restore2EC`.

2.4 Reducing Branch Vertices

In this section, we present an algorithm called `BranchReduction`, that seeks to cut down the

number of branch vertices in a sub-graph. Given a starting 2-edge-connected graph $G = (V, E)$, an edge subset $\bar{E} \subset E$ is considered *redundant* if the graph $G'(V, E \setminus \bar{E})$ is still 2-edge-connected. Note that G' has the same set of edges of G. The aim of the `BranchReduction` algorithm is to identify and remove this unnecessary edge set $\bar{E}$, removing only the ones incident to branch vertices of G. The algorithm is based on the 2-edge-connected subgraph properties that was proposed by Laureana [Lau19]; for easy of read, her claims and proofs are presented below.
Given a starting graph G and a branch vertex v, by removing v from G we obtain a sub-graph $G' = (V', E')$, where the set of vertices is $V' = V \setminus \{v\}$ and the set of edges is $E' = E \setminus \{e : e \in \delta_G(v)\}$, only one of the following three situations can occurs:

(1) G' is not connected;

(2) G' is 2-edge-connected;

(3) G' is connected but not 2-edge-connected.

Since in case (3) the set of bridges $B(G')$ in G' is not empty, by deleting all the bridges from G', we can differentiate two further cases investigating the connected components $C_1, \ldots, C_t$ left in the subgraph of G' induced by the set of edges $E' \setminus B(G')$:

(3a) $|\delta(C_i) \cap B(G'))| \leq 2$, for any $i \in \{1, \ldots, t\}$;

(3b) there exists $i \in \{1, \ldots, t\}$, such that $|\delta(C_i) \cap B(G'))| \geq 3$.

In case (1), v is said to be a cut vertex in G and, by Lemma 1, it is necessarily branch in any feasible solution to the 2ECMBV problem. In case (2), (3a) and (3b), Lemma 2, Lemma 3 and Lemma 4 hold, respectively.

Lemma 1. *Given a 2-edge-connected graph $G = (V, E)$ and a vertex $v \in V$, if v is a cut vertex in G, then it is branch in any 2-edge-connected spanning subgraph of G.*

Proof. $G[V \setminus \{v\}]$ consists of several connected components by definition of cut vertex, $C_1, \ldots, C_\ell$, $\ell \geq 2$. At least two of the edges in $\delta_G(v)$ having one endpoint in C_i belong to any spanning 2-edge-connected subgraph $\bar{G}$ of G, otherwise the subset of vertices $C_i \cup \{v\}$ would not be 2-edge-connected in $\bar{G}$. Since this holds for every $i \in \{1, \ldots, \ell\}$, the number of edges in $\delta_{\bar{G}}(v)$ is at least equal to 2ℓ, which implies that v is a branch vertex in $\bar{G}$. □ □

Lemma 2 ([Lau19])**.** *Given a 2-edge-connected graph $G = (V, E)$ and a branch vertex $v \in V$, if $G[V \setminus \{v\}]$ is 2-edge-connected, then there exists a 2-edge-connected spanning subgraph G_v of G, such that v is not branch in G_v.*

Proof. Since $G[V \setminus \{v\}]$ is 2-edge-connected, the subgraph $G_v = (V, (E \setminus \delta(v)) \cup \{e, f\})$ is 2-edge-connected for any $e, f \in \delta(v)$. Furthermore, v is not branch in G_v, as it has degree two in it. □

Let us denote by G/v the component graph associated to G', defined as follows. The set of vertices of G/v contains a vertex for each connected component C_i in the subgraph obtained by removing from G' the bridges $B(G')$. Furthermore, two vertices of G/v are connected by an edge if and only if the associated components C_i and C_j are connected by a bridge in G'. By construction, the edges incident on C_i are the edges in $\delta(C_i) \cap B(G')$, and G/v is a tree.

Lemma 3 ([Lau19]). *If (3a) holds, then there exists a 2-edge-connected spanning subgraph G_v of G, such that v is not branch in G_v.*

Proof. If (3a) holds, the graph G/v is a path, with exactly two leaves in it, namely there are two connected components C_i and C_j with $i, j \in \{1, \ldots, t\}$, such that $|\delta(C_i) \cap B(G')| = |\delta(C_j) \cap B(G')| = 1$, while $|\delta(C_k) \cap B(G')| = 2$, for any $k \neq i, j$. Since G is 2-edge-connected, there exist $e \in \delta(v) \cap \delta(C_i)$ and $f \in \delta(v) \cap \delta(C_j)$. The subgraph $G_v = (V, E_v)$, where $E_v = E \setminus \delta(v) \cup \{e, f\}$, is 2-edge-connected and v is not branch in it. □

Lemma 4 ([Lau19]). *If (3b) holds, vertex v is branch in any feasible solution to the 2ECMBV problem.*

Proof. When (3b) holds, there is at least one vertex of degree three in the graph G/v, there are at least three leaves, namely there are three connected components C_i, C_j and C_k such that $|\delta(C_i) \cap B(G')| = |\delta(C_j) \cap B(G')| = |\delta(C_k) \cap B(G')| = 1$. Since G is 2-edge-connected, there exist $e \in \delta(v) \cap \delta(C_i)$, $f \in \delta(v) \cap \delta(C_j)$ and $g \in \delta(v) \cap \delta(C_k)$. To ensure 2-edge-connectivity e, f and g must be selected in any feasible solution, thus v is branch in any feasible 2ECMBV solution. □

Algorithm 1 contains the pseudocode of the `BranchReduction` procedure, which takes as input a sub-graph G' of G which is 2-edge-connected, with the aim to reducing the number of branch vertices removing redundant edges. At line 1-4, the procedure processes the branch vertices in G' one by one according to their degree (in ascending order) using a priority queue Q. For each vertex v extracted from the queue, at line 5, the procedure check whether v is still a branch vertex, in this case, at line 6, the algorithm remove v from G' and on line 7 it compute the connected components C of the graph G''. At lines 8-9, if $|C|$ contains more than a single connected component, v is considered as a cut vertex in G' and, by Lemma 1, it is not possible to obtain a 2-edge-connected sub-graph where v is not branch, so we do nothing. Otherwise on line 10, the algorithm compute the bridges B in G''. By Lemma 2, if no bridge exists, we can remove from G' all the edges in $\delta_{G'}(v)$, except for the two edges whose other endpoints have highest degree, this is done on the lines 11-12; in addition to making v not branch, this option may reduce the number of branch vertices in the neighborhood of v. Otherwise on lines 14-21, the algorithm remove existing bridges from G'' and recompute on line 15 the connected components $C_1, C_2, \ldots, C_k$ in G'': for C_i on which a single bridge in B is incident, we remove all the edges in $\delta_{G'}(v)$ except the edge (v, z) which maximizes the degree of z. This implies minimizing the number of edges that are incident on v: when each discovered component has no more than two incident bridges, Lemma 3 holds and only two edges in $\delta_{G'}(v)$ are selected; otherwise, we know from Lemma 4 that v is necessarily a branch vertex. As a result, at least three edges in $\delta G'(v)$

are chosen. For each branch vertex in the input sub-graph G', the procedure executes at most one iteration. The most costly operation in each cycle is computing the connected components of G". A breadth-first search of the sub-graph is used to find connected components, hence in the worst case, the algorithm's time complexity is $O(b(n+m))$, where b is the number of branch vertices in the sub-graph.

Algorithm 1: BranchReduction

Input: A 2-edge-connected graph $G' = (V', E')$

$Q \leftarrow \{v : d_{G'}(v) > 2\}$
while $|Q| > 0$ **do**
  $v \leftarrow v \in Q : d_{G'}(v) \leq d_{G'}(u), \forall u \in Q$
  $Q \leftarrow Q \setminus \{v\}$
  if $d_{G'}(v) > 2$ **then**
    $G'' \leftarrow (V' \setminus \{v\}, E' \setminus \{e : e \in \delta_{G'}(v)\})$
    $C \leftarrow$ connected components of G''
    if $|C| > 1$ **then**
      continue
    $B \leftarrow$ bridges in G''
    if $|C| == 1 \wedge |B| == 0$ **then**
      $G' \leftarrow (V', E'' \cup \arg\max_{(v,x),(v,y) \in \delta_{G'}(v)} \{\min(d_{G''}(x), d_{G''}(y))\})$
    else
      $G'' \leftarrow (V'', E'' \setminus B)$
      $C_1, C_2, \ldots, C_k \leftarrow$ connected components of G''
      $F = \emptyset$
      for $i = 1 \ldots k$ **do**
        if $|\delta_{G'}(C_i) \cap B| == 1$ **then**
          $e = \arg\max_{e=(v,z) \in \delta_{G'}(v)} \{d_{G''}(z)\}$
          $F \leftarrow F \cup \{e\}$
      $G' \leftarrow (V', E'' \cup F)$
  else
    break
return G'

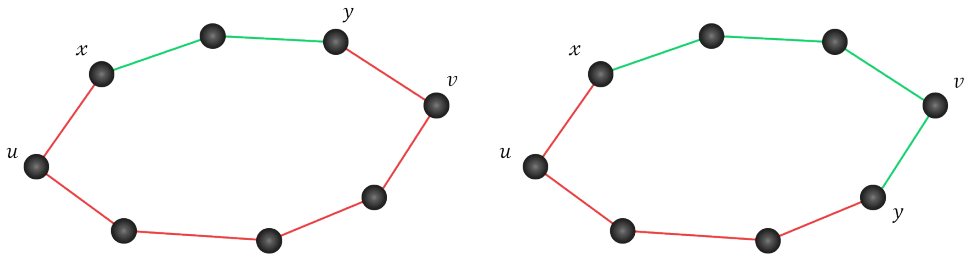
2.5 Finding Feasible Solutions

In this section, we introduce a randomized algorithm called `BuildSolution`, designed to quickly produce diversified solutions for the 2ECMBV problem. Different executions of the algorithm

are likely to produce different solutions. A collection of these solutions supports the generation of a diversified starting population for the GA. The algorithm is founded on the following 2-edge-connectivity property.

Proposition 2.5.1. *Given a graph $G = (V, E)$ and two vertices $u, v \in V : u \neq v$, if u and v are connected by $k \geq 2$ edge-disjoint paths $P_1, P_2, \ldots, P_k$, any pair of vertices in $P_1, P_2, \ldots, P_k$ is connected by at least 2 edge-disjoint paths.*

Proof. Given a graph $G = (V, E)$, $u, v \in V : u \neq v$, and k dge-disjoint paths between u and v ($P = \{P_1, P_2, ..., P_k\}$), with $k \geq 2$. Given a generic vertex x in P_i and a a generic vertex y in P_j, with $P_i, P_j \in P$, the next step is to show that there are at least two edge-disjoint pathways that connect x and y. The following two cases may occur:



(a) (b)

Figure 2.3: (a) In this scenario the vertices x and y are on the same path that connects u and v. (b) Scenario in which they are on different paths. In both (a) and (b) there are 2 edge-disjoint paths between x and y, highlighted by different colors.

- $\mathbf{P_i = P_j}$: In this scenario, the vertices x and y are on the same path and, let's assume that x comes before y, that is P_i has the following structure: $(u, ..., x, ..., y, ..., v)$ (Figure 2.3(a)). A first path $p' : (x, ..., y)$ can be identified between x to y (that are along P_i) in green in Figure 2.3(a). A second path $p'' : (x, ..., u, ..., v, ..., y)$ in red in Figure 2.3(a), where the edges from x to u and from v to y belong to P_i, and the edges from u to v belong to a path $P_t \in P$, with $P_t \neq P_i$, that exists because, by hypothesis, $k \geq 2$. p' and p'' do not share edges.

- $\mathbf{P_i \neq P_j}$: In this scenario, the vertices x and y are located on two different paths, P_i and P_j. The first path between x and y has the form $p' : (x, ..., u, ..., y)$ in red in Figure 2.3(b) and it includes the vertices from x to u along P_i and the vertices from u to y along P_j. The second path between the vertices x and y has the form $p'' : (x, ..., v, ..., y)$ in green in Figure 2.3(b), which contains the vertices from x to v along P_i and the vertices from v to y along P_j. By hypothesis P_i and P_j are edge-disjoint paths, then p' and p'' are edge-disjoint paths, too.

□

By using Proposition 2.5.1, we designed the constructive procedure `BuildSolution` which incrementally builds a set S of vertices inducing a 2-edge-connected subgraph of G, by adding edge-disjoint paths between vertices in S and vertices in $V \setminus S$, until S contains all the vertices of G. The pseudocode of this procedure is given in Algorithm 2.

The algorithm takes as input the original 2-edge-connected graph G. After initializing the output graph G' with an empty set of edges (line 1), a first vertex of G is randomly selected and added to S (line 2). By now, the algorithm adds a vertex $u \in V \setminus S$ to S only if u is 2-edge-connected with some vertex in S. To this end, at each iteration, the following operations are performed: *(i)* vertices x and y are randomly selected from S and $V \setminus S$, respectively (lines 4-5); *(ii)* a path P_1 in G between x and y is found and the edges in P_1 are temporarily removed from G (line 6-7); *(iii)* a second x-y path P_2 in G is found (line 8) and, since the edges of P_1 have been removed from G, P_1 and P_2 are edge-disjoint paths connecting x and y; *(iv)* the edges in P_1 are re-added to G (line 9); *(v)* finally, all the edges in P_1 and P_2 are added to G' and, since we know from Proposition 2.5.1 that every pair of vertices in P_1 and P_2 is 2-edge-connected in G', we add every vertices in P_1 and P_2 to G' (lines 10-11).

Algorithm 2: BuildSolution

Input: The original 2-edge-connected graph $G = (V, E)$
1 $G' \leftarrow (V, E' = \emptyset)$
2 $S \leftarrow \{\text{a random vertex in } V\}$
3 **while** $|V \setminus S| > 0$ **do**
4 $\quad x \leftarrow$ random vertex in S
5 $\quad y \leftarrow$ random vertex in $V \setminus S$
6 $\quad P_1 \leftarrow x$-y path in G
7 $\quad E \leftarrow E \setminus \{\text{edges in } P_1\}$
8 $\quad P_2 \leftarrow x$-y path in G
9 $\quad E' \leftarrow E' \cup \{\text{edges in } P_1\} \cup \{\text{edges in } P_2\}$
10 $\quad S \leftarrow S \cup \{\text{vertices in } P_1\} \cup \{\text{vertices in } P_2\}$
11 **return** G'

When $V \setminus S$ is empty, the algorithm return G'. The complexity of the Algorithm 2 depend on which vertices are randomly chosen during each iteration. In the ideal scenario, only one iteration is necessary, the two computed disjoint paths cover all nodes; however, in the worst scenario, when only one vertex is added to S in every iteration, the main loop takes $n - 1$ iterations, which is $O(n)$. Finding a path between two graph vertices, which is done precisely twice in every iteration, is the most expensive operation. This can be achieved with several algorithms. We adopt a simple visit of the graph, which takes time $O(n+m)$. As a result, in the worst case, the algorithm's complexity is $O(n(n+m))$. Due to the 2-edge-connected nature of the input graph $m > n$, the worst case time complexity is equal to $O(nm)$.

When investigating the complexity of a randomized algorithm, the estimated running time

is typically addressed based on the random decisions made by the algorithm [CLRS22]. In fact, the worst scenario can only be decided by the random selections chosen; it cannot be calculated based on the input. A recurrence relation allows us to describe the time complexity of the algorithm. We establish the following recurrence relation by denoting with $T(n)$ the quantity of iterations required to construct a solution for a network with n vertices (where $S^{(i)}$ denotes set S at iteration i, with $|V \setminus S^{(0)}| = n$.):

$$T(|V \setminus S^{(i)}|) = 2 \cdot O(n+m) + T(|V \setminus S^{(i+1)}|),$$

The total number of operations performed by the algorithm to construct a solution for an instance of size n are: *i)* $2 \cdot O(n+m)$ operations to obtain two edge-disjoint paths between a vertex in S and one in $V \setminus S(i)$ in the starting graph G; *ii)* the operations required to complete the next iterations depends on the number of vertices in $V \setminus S^{(i+1)}$. The set $V \setminus S(i)$ is split into $V \setminus S(i+1)$ and $(V \setminus S(i)) \setminus (V \setminus S(i+1))$ at each step. The running time recurrence is impacted by the correlation of this partitioning. A single iteration is sufficient if the best case partitioning occurs, which is when $|V \setminus S(1)| = \emptyset$, and the recurrence has the solution in $O(m+n)$ time. On the contrary, if the partitioning in every step is unbalanced (it adds in each iteration only one node in S), i.e. $|V \setminus S^{(i+1)}| = |V \setminus S^{(i)}| - 1 \ \forall i$, the recurrence require $O(n(m+n)) = O(nm)$ time in order to obtain a solution, which correspond to the worst case. Finally, the `BuildSolution` procedure has an estimated running time of $O\left((n+m)\log_2 n\right)$ in the average case on random 2-edge-connected graphs.
A logarithmic number of steps is needed to solve the resulting recurrence, when after each step of the recurrence relation, the number of vertices remaining in $V \setminus S^{(i+1)}$ is a fraction of n. The only prerequisite is that we do not remove a predetermined number of vertices from $V \setminus S(i)$ at each step i, regardless of how small this fraction is. We indicate with $\frac{n}{\alpha^i}$ the number of vertices remaining in $V \setminus S^{(i+1)}$ after each iteration i, with $\alpha > 1$ and $\alpha \in \mathbb{R}$.

$$\begin{aligned} T(|V \setminus S^{(0)}|) &= 2c(n+m) + T\left(\frac{n}{\alpha}\right) && (2.1)\\ &= 2c(n+m) + 2c\,(n+m) + T\left(\frac{n}{\alpha^2}\right) && (2.2)\\ &= 2c(n+m) + 2c\,(n+m) + \cdots + 2c\,(n+m) + T\left(\frac{n}{\alpha^j}\right), && (2.3) \end{aligned}$$

Where j is the number of iterations required to solve the relation and $S(0) = \emptyset$. To obtain the value of j we put n/α^j equal to 1, thus $j = \log_\alpha n = \log_2 n / \log_2 \alpha = O(\log_2 n)$. While it is obvious that the partitioning does not always result in divides with constant proportionality at each stage, but sufficiently well balanced and fairly unbalanced splits are expected to be randomly distributed in the recursion tree. It has been demonstrated that even in this scenario, $O(\log_2 n)$ is the anticipated number of rounds until a best-case partitioning occurs [CLRS22].

2.6 Genetic Algorithm

In this section, we describe our resolution approach for the 2ECMBV problem which is a genetic algorithm. General aspects of Genetic Algorithms can be found in Section A.3.2 of Appendix A.

2.6.1 Chromosome representation and fitness function

In our algorithm, each chromosome $\mathcal{C}$ is associated to a feasible solution of the 2ECMBV problem on G. For this reason, $\mathcal{C}$ is a binary vector whose size is equal to the number of edges of G and the ith gene in it is equal to 1 if the edge e_i of G belongs to the solution and zero otherwise. We denote by $\mathcal{C}[i]$ the value of ith gene of chromosome $\mathcal{C}$.

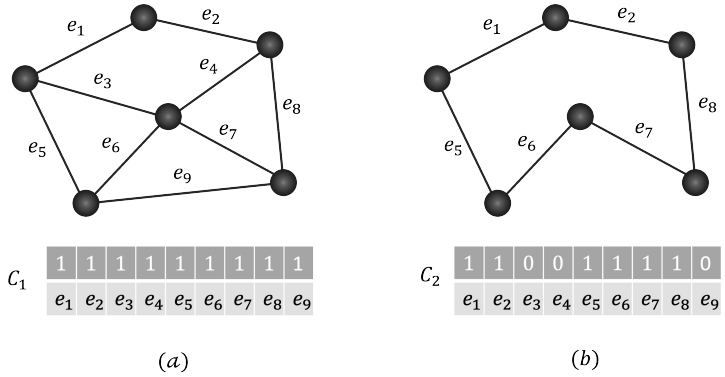


Figure 2.4: (a) The original graph G and (b) a 2-edge-connected subgraph of G with their corresponding encodings, C_1 and C_2, respectively.

An example of a feasible solution encoding is shown in Figure 2.4. Here $\mathcal{C}_1[i] = 1$ for all $i = 1, \ldots, 9$ while $\mathcal{C}_2[i] = 1$, for i=$\{1,2,5,6,7,8\}$ and $\mathcal{C}_2[i] = 0$, for i=$\{3,4,9\}$. From now on, we denote by $G(\mathcal{C})$ the subgraph of G induced by selected edges in $\mathcal{C}$. According to this definition, $G(\mathcal{C}_2)$ is the subgraph of G depicted in Figure 2.4(b).

The chromosomes of the population are ranked according to a *fitness function* $\mathcal{F}$. In our algorithm, the fitness function $\mathcal{F}(\mathcal{C})$ of a chromosome $\mathcal{C}$ is equal to the number of branch vertices in $G(\mathcal{C})$. The lower is the number of branch vertices the better is the fitness value. In Figure 2.4, we have that $\mathcal{F}(\mathcal{C}_1) = 5$ while $\mathcal{F}(\mathcal{C}_2) = 0$ and then $G(\mathcal{C}_2)$ is a better solution than $G(\mathcal{C}_1)$.

2.6.2 Initial population

The initial population is generated by using the `BuildSolution` procedure described in Section 2.5. Thanks to its randomized nature, the algorithm is able to generate, from the same input graph, different and heterogeneous feasible solutions. We repeatedly invoke this procedure until a fixed number of individuals, equal to the size $\mathcal{N}$ of the population, is obtained.

2.6.3 Selection, Crossover and Mutation operators

The selection of two parents for the reproduction is carried out by using the *Tournament Selection* policy. This technique consists of randomly choosing t chromosomes from the current population and then the one with the best fitness function value is chosen as first parent. The process is iterated to select the second parent. In our implementation, t is equal to 3.

Two *crossover* operators, both taking as input two parent chromosomes and generating a new child chromosome, were designed: the former focuses on feasibility and naturally guarantees the 2-edge-connectivity and spanning properties, while the latter tries to reduce the number of branch vertices and exploits the restorer operator to assure that the child chromosome is a feasible solution. More in detail, the first crossover operator generates the child chromosome by copying one of two parents and then by adding to it some edges of the other parent. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the two parent chromosomes and let $G(\mathcal{C}_1) = (V_{\mathcal{C}_1}, E_{\mathcal{C}_1})$ and $G(\mathcal{C}_2) = (V_{\mathcal{C}_2}, E_{\mathcal{C}_2})$ be the two induced subgraphs. To produce the child chromosome $\mathcal{C}_c$, the crossover operator randomly select one of two parents, w.l.o.g. let us suppose $\mathcal{C}_1$, and makes $\mathcal{C}_c$ a copy of this parent. Then, each edge in $E_{\mathcal{C}_2}$ is added to $E(\mathcal{C}_c)$, i.e., the related gene is set to 1 in $\mathcal{C}_c$, with probability 0.5. Figure 2.5 shows how this crossover works. As said, this first operator focuses on feasibility, indeed it always produces feasible solutions whose values are worse than or equal to those of the reference parents: the improvement of such values is left to the next operators. On the other hand, the second crossover operator generates the child chromosome $\mathcal{C}_C$ by performing the following three steps: *(i)* initially, all the edges in $E_{\mathcal{C}_1} \cap E_{\mathcal{C}_2}$ are added to $E_{\mathcal{C}_C}$; then *(ii)* the edges incident on branch vertices in $G(\mathcal{C}_C)$ are removed; and finally the restorer operator is invoked on $G(\mathcal{C}_C)$. Figure 2.6 depicts an example of this crossover. Considering the edges from both the parents which do not lead to branch vertices and then completing the solution is more expensive, but is more likely to produce promising child chromosomes. In our implementation, the two designed crossover operators are performed alternatively, with probabilities of 30% and 70%, respectively.

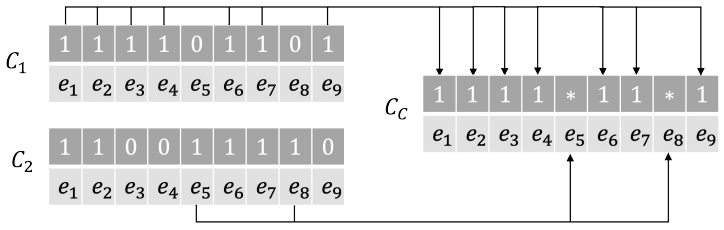


Figure 2.5: First crossover: the arrows represent how parents' genes influence those of their children $\mathcal{C}_C$. The special character $*$ means random choice between 0 and 1.

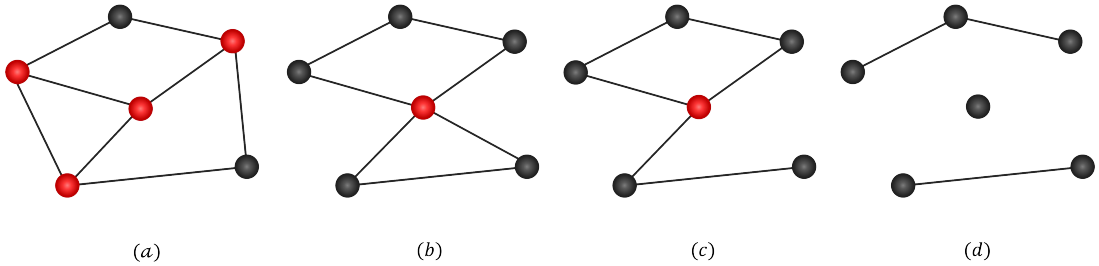


Figure 2.6: Second crossover. (a) The first parent $G(\mathcal{C}_1)$. (b) The second parent $G(\mathcal{C}_2)$. (c) The subgraph induced by the set of common edges $E_{\mathcal{C}_1} \cap E_{\mathcal{C}_2}$. (d) The subgraph remaining after removing the edges incident on branch vertices. The child chromosome $\mathcal{C}_C$ is obtained by executing the `Restore2EC`operator on the last graph.

Generally, in genetic algorithms, after the generation of a new individual from two parents, a *mutation* operation is invoked with a certain probability. The goal is to introduce new genetic material into the population allowing to consider unexplored areas of the feasible region and thus reduce the probability to be trapped in local optima. Let's denote with $1_{\mathcal{C}}$ and $0_{\mathcal{C}}$ respectively the number of genes equal to one and equal to zero in $\mathcal{C}$. In our GA, the mutation operation, performed with probability $\mathcal{P}_m$, randomly chooses $\lceil \mathcal{J}_m \cdot 1_{\mathcal{C}} \rceil$ bits equal to one and $\lceil \mathcal{J}_m \cdot 0_{\mathcal{C}} \rceil$ bits equal to zero and inverts them, where $\mathcal{J}_m \in [0, 1]$ is the mutation impact parameter, indicates the percentage of these bits that must be inverted.

2.6.4 Local Search

After the application of the crossover and mutation operators, the new individual can contain much more branch vertices than its parents. For this reason, we implemented a `LocalSearch` procedure which aims to reduce the number of these branch vertices by performing a sequence of edge replacements.

The idea behind the `LocalSearch` approach is the following. Given the current solution $G(\mathcal{C})$, let u and v be two branch vertices in $G(\mathcal{C})$; The local search removes an edge $(u, w) \in \delta_{G(\mathcal{C})}(u)$ and inserts in $G(\mathcal{C})$ a new edge $(w, v) \in \delta_G(w) \setminus \delta_{G(\mathcal{C})}(w)$, currently not in $\mathcal{C}$, to try to preserve the the 2-edge-connectivity. As consequence, the degree of u in $G(\mathcal{C})$ is decreased by one while the degree of v is increase by one. thanks to these replacements, the aim of `LocalSearch` is to remove the branch vertices from $G(\mathcal{C})$ by reducing their degree to 2. This type of operation cannot assure the 2-edge-connectivity of the new solution so, as last step, `LocalSearch` invokes the `Restore2EC` procedure in order to restore the 2-edge-connectivity if this had been lost.

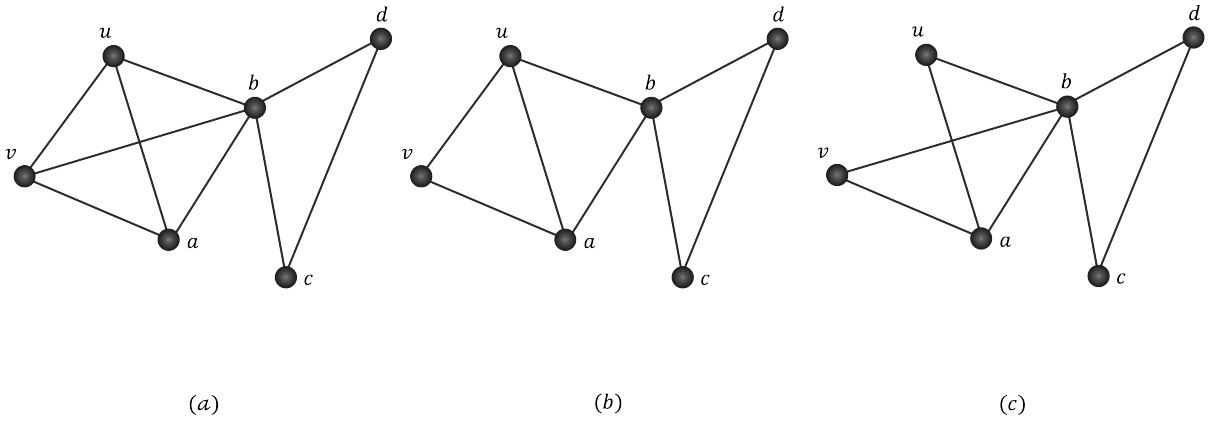


Figure 2.7: (a) A starting graph G with four branch vertices u, b, a and v; (b) a sub-graph G' of G with three branch vertices: u, b and a; (c) a sub-graph G'' of G with two branch vertices, a and b, obtained from G' by replacing the edge (v, u) with the edge (v, b).

For instance, let us consider the graph $G = (V, E)$ shown in 2.7(a) and the 2-edge connected subgraph $G' = (V', E')$ in 2.7(b) with three branch vertices a, b and u. By invoking `LocalSearch` procedure on G', it removes from the branch vertex u the edge $(u, v) \in E'$ and adds to the branch vertex b the edge $(v, b) \in E \setminus E'$ (Figure 2.7(c)). In this way, b remains a branch vertex while u is no longer a branch vertex. It is easy to see that, according to this strategy, the `LocalSearch` procedure never introduces new branch vertices but it could reduce their number.

The pseudocode of the `LocalSearch` procedure is reported in Algorithm 3.

Given the graph G and the chromosome $\mathcal{C}$, the procedure sets $G(\mathcal{C})$ as the subgraph of G associated to $\mathcal{C}$ and sets B to the empty set (line 1-2). All the branch vertices in $G(\mathcal{C})$ are then added to B (line 3-4). The while loop, in line 5, iterates until the set B gets empty. In this loop a branch vertex b of B is randomly selected (line 6) and for each edge (b, v) incident to b that is not in $E_{\mathcal{C}}$, the procedure looks for another edge (v, u) in $E_{\mathcal{C}}$ having u as branch vertex (line 10). If this last edge is found, `LocalSearch` adds (b, v) to $E_{\mathcal{C}}$, removes (v, u) from $E_{\mathcal{C}}$ and stops the for loop of line 9 (lines 11-12). In order to increase the possibility to preserve the 2-edge-connectivity while keeping the fitness value unchanged, in line 13 the procedure adds to $E_{\mathcal{C}}$ all the edges in $E \setminus E_{\mathcal{C}}$ which endpoints are both branch vertices. Finally, `LocalSearch` invokes `Restore2EC` to assure that the solution returned satisfy the 2-edge-connectivity constraint.

2.6.5 Shaking and Cleaning

A shaking operator has been implemented to escape from the local optimum and to encourage a wider exploration of the solution space. After $\mathcal{I}_{\text{SH}}$ iterations of the genetic algorithm without improvements of the incumbent solution, $\lfloor \mathcal{N} \cdot \mathcal{J}_{\text{SH}} \rfloor$ random individuals are replaced with new individuals generated from scratch. Here, $\mathcal{J}_{\text{SH}} \in [0, 1]$ is the shaking impact parameter, which determines the number of individuals of the population to be replaced.

Algorithm 3: LocalSearch

Input: G and $\mathcal{C}$

```
G(C) = (V_C, E_C)   //subgraph of G associated to C
B ← ∅
foreach vertex u ∈ V_C do
    if d_{G(C)}(u) > 2 then B ← B ∪ {u}
while B is not empty do
    b ← random branch vertex in B
    for (b, v) ∈ δ_G(b) do
        if (b, v) ∉ E_C then
            for (v, u) ∈ δ_{G(C)}(v) do
                if d_{G(C)}(u) > 2 then
                    E_C ← E_C \ {(v, u)} ∪ {(b, v)}
                    break
add in E_C all (u, v) edges with u and v branch.
return Restore2EC(G, G(C))
```

When the genetic algorithm ends its execution, a cleaning policy is applied by carrying out the `BranchReduction` procedure on all the individuals of the final population. As previously described, this allows to decrease the number of branch vertices by removing a large number of redundant edges.

2.6.6 Termination criteria

GA iterates until one of the two following stopping criteria is reached.

- The first criterion is based on $\mathcal{I}_{\text{MAX}}$ parameter, representing the maximum number of iterations that the algorithm can carry out;
- The second criterion is based on the $\mathcal{I}_{\text{SH}}$ parameter: if, after the application of the shaking operator, a new sequence of $\mathcal{I}_{\text{SH}}$ iterations not improving the best chromosome fitness value occur, the algorithm stops.

2.7 Computational Tests

In this section we present the computational results of the tests we made in order to evaluate the effectiveness and the performance of GA. The algorithm was coded in Python by using *NetworkX*

[HSS08] library on a Linux platform running on an Intel Xeon E5 2.3 GHz processor with 128 GB of RAM.

In the following two subsections, we describe how we generated the instances for the 2ECMBV and how the parameters of GA were chosen. The computational results are reported in subsection 2.7.3.

2.7.1 Test instances

To the best of our knowledge, the only instances available in the literature for the 2ECMBV problem are the ones presented in [Lau19]. However, since these instances were generated to test the B&C algorithm, presented in that work, they are too small to be used to test the effectiveness and performance of a metaheuristic. For this reason, we extended this dataset with new larger instances generated by using the same strategy proposed in [Lau19]. The idea behind the instance generation procedure is to obtain a non-Hamiltonian and 3-connected graph, ensuring that there are not essential edges or vertices which must necessarily be branch.

More in detail, given a starting clique graph $G' = (V', E')$ with $|V'| = n' \geq 4$, let q be an integer such that $n' \geq 3q$. The procedure defines q disjoint subsets $W_1, \ldots, W_q$ of V' with $|W_i| = 3, i = 1 \ldots, q$. Then it generates q disjoint sets, $T_1, \ldots, T_q$, of new vertices, not in V', with $|T_i| \geq 3, i = 1, \ldots, q$. Finally, the procedure build the graph $G = (V, E)$ with $V = V' \cup \bigcup_{i=1}^{q} T_i$ and $E = E' \cup \bigcup_{i=1}^{q} \{(u, v) : u \in T_i, v \in W_i\}$. The graph $G = (V, E)$ just built is 3-connected and non-Hamiltonian (the proof is provided in [Lau19]).

Note that $|V| = n' - \bar{n}$ where $\bar{n} = \sum_{i=1}^{q} |T_i|$ is the number of vertices that are added to the starting graph G'. In our computational tests, we grouped the instances in two sets: the *Small instances* where $n' \in \{20, 30, 40, 50\}$, which are the same used in [Lau19] and the *Large instances* where $n' \in \{60, 70, 80, 90, 100\}$. For each combination of n', $\bar{n}$, and q, five different instances were generated that together represent a scenario. Thus, in total, this first set consists of 630 individual instances, grouped in 126 scenarios.

To further investigate the effectiveness of GA, we generated a set of random Hamiltonian instances. Since the optimal solution of 2ECMBV is always zero in these instances, then we can compare the solutions found by GA with the optimal ones. The generation of the instances is carried out as follows. Given n vertices and a probability d, we first create a Hamiltonian cycle randomly and then, for every couple of vertices i and j we introduce the edge (i, j) in the graph with probability d. For the generation of the instances we used the following values: $n \in \{100, 150, 200, 250, 300, 350, 400\}$ and $d \in \{0.3, 0.5, 0.7\}$. For each combination of these two parameters, we generated five different instances for a total of 105 Hamiltonian instances grouped in 21 scenarios.

2.7.2 Parameters tuning

To find the best performing values for the parameters of the GA, we used the *IRACE* package [LIDLPC+16], an automatic configuration tool for parameter setting. IRACE was executed on a subset of 147 instances selected according to all the possible combinations of the n and

m parameters values. In particular, 126 instances were selected from the non-Hamiltonian and 3-connected set, while the remaining 21 instances were selected from the Hamiltonian set. Table 2.1 reports, for each parameter, the set of tested values (*Values*) and the corresponding target value (*Target*) in the best configuration found by IRACE.

Parameter	**Values**	**Target**
$\mathcal{N}$	$\{30, 50, 70\}$	30
$\mathcal{I}_{\text{MAX}}$	$\{30, 50, 70\}$	50
$\mathcal{P}_m$	$\{0.05, 0.1, 0.15\}$	0.1
$\mathcal{J}_m$	$\{0.02, 0.04\}$	0.02
$\mathcal{I}_{\text{SH}}$	$\{3, 5, 7\}$	7
$\mathcal{J}_{\text{SH}}$	$\{1, 1/3, 1/4\}$	$1/3$

Table 2.1: Tested sets of values and *IRACE* choices for GA parameters.

2.7.3 Computational results

In this section, we verify the effectiveness of GA algorithm by comparing it with the B&C proposed in [Lau19]. The two algorithms are executed on the same machine and for the B&C we set a time limit equal to 2 hours.

					B&C		GA		
n′	**n̄**	**q**	**n**	**m**	**Obj**	**Time**	**Obj**	**Time**	**Gap**
20	10	2	30	220	3.40	0.04	3.40	10.50	0.0
		3	30	220	3.00	0.06	3.00	9.56	0.0
	16	3	36	238	4.40	0.07	4.40	12.11	0.0
		5	36	238	5.00	2.60	5.00	9.68	0.0
	20	4	40	250	6.00	0.12	6.00	12.67	0.0
		6	40	250	7.00	12.55	7.00	10.19	0.0
	30	4	50	280	7.20	0.21	7.20	17.85	0.0
		6	50	280	8.60	2.33	8.60	14.26	0.0
	40	4	60	310	6.60	0.23	6.60	24.38	0.0
		6	60	310	8.40	1.47	8.40	19.99	0.0
	50	4	70	340	7.80	0.40	7.80	33.32	0.0
		6	70	340	10.40	5.92	10.40	26.91	0.0
	60	4	80	370	7.40	0.33	7.40	42.35	0.0
		6	80	370	10.40	2.61	10.40	33.41	0.0
30	15	3	45	480	4.40	0.17	4.40	22.35	0.0
		5	45	480	5.00	2.13	5.00	18.04	0.0
	24	4	54	507	6.60	0.34	6.60	24.94	0.0
		8	54	507	8.00	25.43	8.00	19.29	0.0
	30	6	60	525	8.40	7.15	8.40	24.06	0.0
		10	60	525	10.00	1030.81	10.00	19.30	0.0
	45	6	75	570	10.40	2.72	10.40	36.49	0.0
		10	75	570	11.00	605.38	11.00	30.54	0.0
	60	6	90	615	10.40	8.94	10.40	49.90	0.0
		10	90	615	15.40	248.00	15.40	37.15	0.0
	75	6	105	660	11.60	9.97	11.60	67.43	0.0
		10	105	660	15.40	646.42	15.40	50.79	0.0
	90	6	120	705	11.20	11.91	11.20	84.79	0.0
		10	120	705	18.20	617.87	18.20	65.67	0.0
40	20	4	60	840	7.00	1.73	7.00	37.00	0.0
		6	60	840	7.00	11.34	7.00	32.43	0.0
	32	6	72	876	7.60	10.75	7.60	39.98	0.0
		10	72	876	11.00*	7201.41	11.00	32.33	0.0
	40	8	80	900	11.40	509.73	11.40	43.49	0.0
		13	80	900	13.00*	7215.75	13.00	34.25	0.0
	60	8	100	960	13.60	111.74	13.60	58.73	0.0
		13	100	960	14.00*	7214.94	14.00	59.44	0.0
	80	8	120	1020	13.40	173.60	13.40	80.98	0.0
		13	120	1020	20.20*	4262.95	20.20	61.63	0.0
	100	8	140	1080	13.80	211.74	13.80	109.36	0.0
		13	140	1080	22.40*	6289.50	22.40	84.20	0.0
	120	8	160	1140	14.20	149.54	14.20	148.16	0.0
		13	160	1140	22.60*	3559.44	22.60	111.85	0.0
50	25	5	75	1300	6.00	2.84	6.00	58.27	0.0
		8	75	1300	8.00	364.32	8.00	52.12	0.0
	40	8	90	1345	9.00	713.76	9.00	61.05	0.0
		13	90	1345	13.00*	6810.18	13.00	57.51	0.0
	50	10	100	1375	11.00*	2492.98	11.00	72.03	0.0
		16	100	1375	17.00*	7218.56	17.20	58.70	0.2
	75	10	125	1450	11.00*	3271.07	11.00	101.91	0.0
		16	125	1450	17.00*	7218.50	17.00	78.62	0.0
	100	10	150	1525	11.00*	4580.52	11.00	132.02	0.0
		16	150	1525	17.00*	7218.60	17.00	116.93	0.0
	125	10	175	1600	11.00*	2496.49	11.00	191.37	0.0
		16	175	1600	17.00*	7218.80	17.00	156.33	0.0
	150	10	200	1675	11.00*	1557.97	11.00	258.74	0.0
		16	200	1675	17.00*	7218.64	17.00	211.11	0.0
Avg					10.87	1759.89	10.88	60.15	0.004
#Best									55

Table 2.2: Comparison between the solutions of B&C and GA on the Small instances.

Table 2.2 reports the results of GA and B&C on the Small instances. The first five columns show the characteristics of the instances: the number n' of vertices of the graph G', the cardinality $\bar{n}$ of $T_1 \cup \cdots \cup T_q$, the value q, the number n of vertices and the number m of edges of G.

The next four columns report the solution value (Obj) and the computational time ($Time$), in seconds, of B&C and GA, respectively. Whenever B&C reaches the time limit of two hours, the related solution value is marked with a "*" symbol to highlight that this value is an upper bound of the optimal solution value. Each row in the tables represents a scenario composed of five instances with the same characteristics but different seed (used to initialize the random number generator), and the results shown in each line are the average values of these five instances. The last column shows the gap (Gap) between the solution found by GA and the best/optimal solutions. This gap is computed by using the formula: $Gap = Obj(GA) - Obj(B\&C)$. Finally, at the bottom of the table, the Avg row reports the average values of Obj, $Time$ and Gap while the $\#Best$ row shows how many times GA finds the best/optimal solution.

The results of Table 2.2 show that B&C provides the optimal solution on 39 out of 56 scenarios with an average time equal to 1759 seconds. On 55 out of 56 scenarios, GA returns the same solutions of B&C and, in particular, it always finds the optimal solution in the scenarios where this solution is known. In the remaining scenario (50-50-16) the gap from the optimal/best solution is equal to 0.2. The results of the Avg row show that the difference between the average Obj values of the two algorithms is equal to 0.01. Moreover, the average computational time of GA in the Small instances is equal to 60.15 seconds, and it never exceeds the 3 minutes in this set of instances. However, it is worth noting that there are several scenarios where GA is slower than B&C. This occurs because, despite the stopping criteria used by GA, it has to always carry out a minimum number of iterations before stopping, even when it finds the best solution at the first iteration.

					B&C		GA		
n′	**n̄**	**q**	**n**	**m**	**Obj**	**Time**	**Obj**	**Time**	**Gap**
60	30	6	90	1860	8.20	40.05	8.20	84.16	0.0
		10	90	1860	10.00	1169.83	10.00	68.67	0.0
	48	9	108	1914	13.60	727.07	13.60	87.42	0.0
		16	108	1914	16.00*	7218.46	16.00	67.82	0.0
	60	12	120	1950	16.60*	7217.62	16.60	101.83	0.0
		20	120	1950	20.00*	7219.04	20.00	69.26	0.0
	90	12	150	2040	19.40*	5893.34	19.40	122.03	0.0
		20	150	2040	21.00*	7218.70	21.20	117.73	0.2
	120	12	180	2130	21.20*	5057.12	21.20	177.71	0.0
		20	180	2130	30.80*	7211.34	30.80	129.99	0.0
	150	12	210	2220	21.20*	5643.38	21.20	236.21	0.0
		20	210	2220	34.20*	7201.44	34.20	173.92	0.0
	180	12	240	2310	22.60*	4318.54	22.60	320.99	0.0
		20	240	2310	35.00*	7199.04	35.00	228.70	0.0
70	35	7	105	2520	8.00	106.44	8.00	113.83	0.0
		11	105	2520	12.00*	1834.72	12.00	106.95	0.0
	56	11	126	2583	12.00	823.36	12.00	130.54	0.0
		18	126	2583	19.00*	7219.10	19.00	114.08	0.0
	70	14	140	2625	15.00*	6739.40	15.00	137.69	0.0
		23	140	2625	23.00*	7213.12	23.00	129.46	0.0
	105	14	175	2730	15.00*	7208.77	15.00	191.50	0.0
		23	175	2730	24.00*	7209.88	24.00	174.34	0.0
	140	14	210	2835	15.00*	7215.04	15.00	271.18	0.0
		23	210	2835	24.00*	7215.46	24.00	244.17	0.0
	175	14	245	2940	15.00*	5266.06	15.00	418.21	0.0
		23	245	2940	24.00*	7212.86	24.00	292.64	0.0
	210	14	280	3045	15.00*	7214.44	15.00	532.53	0.0
		23	280	3045	24.00*	7214.29	24.00	406.82	0.0
80	40	8	120	3280	9.00*	2052.80	9.00	140.54	0.0
		13	120	3280	13.00*	7217.67	13.00	144.72	0.0
	64	12	144	3352	13.00*	5661.32	13.00	167.71	0.0
		21	144	3352	21.00*	7217.17	21.00	148.49	0.0
	80	16	160	3400	17.00*	7218.16	17.00	171.42	0.0
		26	160	3400	27.00*	7218.77	27.00	149.07	0.0
	120	16	200	3520	17.00*	7219.19	17.00	281.02	0.0
		26	200	3520	27.00*	7219.76	27.00	205.11	0.0
	160	16	240	3640	17.00*	7219.22	17.00	374.10	0.0
		26	240	3640	27.00*	7219.73	27.00	329.79	0.0
	200	16	280	3760	17.00*	7219.35	17.00	605.52	0.0
		26	280	3760	27.00*	7220.03	27.00	420.54	0.0
	240	16	320	3880	17.00*	7219.19	17.00	672.45	0.0
		26	320	3880	27.00*	7219.82	27.00	585.43	0.0
90	45	9	135	4140	10.00*	1595.33	10.00	199.60	0.0
		15	135	4140	15.00*	5689.53	15.00	171.51	0.0
	72	14	162	4221	15.00*	6113.03	15.00	208.69	0.0
		24	162	4221	24.00*	7219.43	24.00	163.24	0.0
	90	18	180	4275	19.00*	7212.73	19.00	218.79	0.0
		30	180	4275	30.00*	7212.53	30.00	184.76	0.0
	135	18	225	4410	19.00*	7212.18	19.00	337.66	0.0
		30	225	4410	31.00*	7212.78	31.00	297.96	0.0
	180	18	270	4545	19.00*	7212.54	19.00	537.10	0.0
		30	270	4545	31.00*	7212.27	31.00	385.45	0.0
	225	18	315	4680	19.00*	7213.94	19.00	736.06	0.0
		30	315	4680	31.00*	7214.27	31.00	565.67	0.0
	270	18	360	4815	19.00*	7215.76	19.00	857.13	0.0
		30	360	4815	31.00*	7216.75	31.00	750.56	0.0
100	50	10	150	5100	11.00*	5003.63	11.00	222.28	0.0
		16	150	5100	17.00*	7218.58	17.00	210.87	0.0
	80	16	180	5190	17.00*	7219.34	17.20	254.29	0.2
		26	180	5190	27.00*	7219.55	27.20	219.37	0.2
	100	20	200	5250	21.00*	7214.37	21.00	296.72	0.0
		33	200	5250	33.00*	7214.10	33.00	238.51	0.0
	150	20	250	5400	21.00*	7209.18	21.00	423.98	0.0
		33	250	5400	34.00*	7209.86	34.20	325.13	0.2
	200	20	300	5550	21.00*	7210.20	21.00	582.60	0.0
		33	300	5550	34.00*	7210.07	34.00	481.60	0.0
	250	20	350	5700	21.00*	7214.75	21.00	955.59	0.0
		33	350	5700	34.00*	7215.47	34.20	684.00	0.2
	300	20	400	5850	21.00*	7212.96	21.00	1169.45	0.0
		33	400	5850	34.00*	7213.10	34.00	1013.71	0.0
Avg					21.00	6270.12	21.01	319.12	0.014
#Best									65

Table 2.3: Comparison between the solutions of B&C and GA on the Large instances.

Table 2.3 shows the computational results of the two algorithms on the Large instances. Table headings have the same meaning that they have for Table 2.2. The results show that B&C provides the optimal solution only in 5 scenarios, while, on the remaining ones, it reaches the time limit. GA finds the same solutions of B&C on 65 out of 70 scenarios and, in the remaining five scenarios, its gap is always equal to 0.2. Moreover, the average gap is very low (0.014). These results highlight the effectiveness of GA which very often finds the best/optimal solution and when this does not occur, its gap from this solution is very low. Regarding the computational time, GA requires on average 319 seconds and never exceeds 1170 seconds. As expected, by increasing the size of the instances the computational time of the two algorithms

is not comparable anymore.

Since in the scenarios where B&C reaches the time limit GA never finds a better solution, even in the largest instances, we suspect that the solutions provided by B&C could be the optimal ones or very close to the optimal ones but the algorithm fails to certify this optimality within the time limit.

As described in Section 2.7.1, we generated a new random set of instances to further investigate the effectiveness of our algorithm. To this end, we have chosen to generate Hamiltonian instances to know a priori the optimal solution value that is zero. Table 2.4 reports the results of GA on this new set of instances.

The first three columns show the characteristics of the instance: the number n of vertices of the graph, the probability d used to generate the edges (see Section 2.7.1) and the number of edges m. This last column reports an interval of values representing the minimum and the maximum number of edges contained inside the graphs of that scenario. Indeed, each line in the tables represents a scenario composed of five instances with the same characteristics but a different seed, and the results shown in each line are the average values of these five instances. The remaining three columns show the optimal solution value (Opt), the solution value (Obj) and the computational time ($Time$), in seconds, of GA, respectively. Notice that we do not report the column Gap in this table because, having an optimal solution value always equal to zero, this column coincides with the column Obj. Finally, at the bottom of the table, the Avg row reports the average values of Obj and $Time$.

				GA	
n	**d**	**m**	**Opt**	**Obj**	**Time**
100	0.3	[1554 - 1615]	0.00	2.20	91.19
100	0.5	[2494 - 2550]	0.00	1.80	111.40
100	0.7	[3464 - 3566]	0.00	1.40	133.46
150	0.3	[3493 - 3587]	0.00	2.60	192.94
150	0.5	[5694 - 5747]	0.00	2.60	264.18
150	0.7	[7846 - 7935]	0.00	1.60	334.73
200	0.3	[6174 - 6307]	0.00	3.00	356.46
200	0.5	[10155 - 10262]	0.00	2.60	482.03
200	0.7	[13970 - 14134]	0.00	2.00	607.44
250	0.3	[9631 - 9779]	0.00	3.80	435.34
250	0.5	[15818 - 16026]	0.00	3.20	749.63
250	0.7	[21870 - 22083]	0.00	2.00	1106.88
300	0.3	[13753 - 14135]	0.00	3.20	763.16
300	0.5	[22649 - 22852]	0.00	3.00	1135.25
300	0.7	[31517 - 31622]	0.00	2.40	1552.63
350	0.3	[18838 - 19004]	0.00	3.20	1168.42
350	0.5	[30846 - 31038]	0.00	3.00	1749.42
350	0.7	[42757 - 42932]	0.00	2.60	2128.46
400	0.3	[24630 - 24776]	0.00	4.20	1457.40
400	0.5	[40340 - 40678]	0.00	3.80	2032.20
400	0.7	[56119 - 56498]	0.00	3.20	2646.19
Avg				2.73	928.51

Table 2.4: Computational results for the Hamiltonian instances.

The results of Table 2.4 show that these new instances are harder to solve for GA. Indeed, the algorithm does not find the optimal solution in these instances. However, the average gap value from the optimal solution is equal to 2.73 and only in one case this gap is over 4. From these results, we derive that GA remains effective even on this type of instance but less effective if compared with the results obtained on the previous benchmark instances. It is worth noting that as the density of the instances increases as the solution quality, provided by GA, improves. For instance, in scenario 100-0.3 the gap is equal to 2.20 while in scenario 100-0.7 this gap decreases to 1.40. We suspect that this occurs because, by increasing the number of edges inside the graph, we increase the chance to have more Hamiltonian cycles and then more optimal solutions. Even the computational time increases in these instances with an average value equal to 928.51 seconds and a peak equal to 2646.19 seconds in the largest scenario 400-0.7. Overall, GA remains sufficiently fast despite the density of these graphs.

Chapter 3

Collapsed k-Core Problem

A problem of fundamental importance today, which concerns the identification of substructures in graphs, is the problem of the Collapsed k-Core. The study of this problem is linked to the great diffusion of social networks and to the attention that today we have towards the so-called influential users.The idea is that the decision of a user to leaving or remaining in the network, is influenced by that of her connected friends. A popular assumption is that a user remains in the network if she has at least a certain number of connections, say k, in the network [BKL+15]. On the contrary, a user is driven to leave the social network if she has less than k connections. Given this assumption, if a user leaves the network, the degree of her neighbors decreases by one, eventually becoming smaller than k for some of them. Thus, a cascade phenomenon is observed each time a user drops out from the network, until a *stable* configuration is obtained, which corresponds to the k-core of the social network graph. In this context, we study the Collapsed k-Core Problem, which has been introduced in [ZZQ+17] to identify the critical users to be eventually incentivized not to leave the network. This problem indeed consists in finding the set of a given number b of users, the exit of whom from the network minimizes the number of the remaining users in the network itself, i.e., leads to the minimal k-core. First of all, we propose a formulation of the problem modeling the cascade effect which determines the withdrawal of a certain number of users from the network. In such approach, a time index is used to represent the subsequent deletion rounds of this process. Beyond that, the tools of bilevel optimization have been recently used for developing exact methods for several critical node/edge detection problems [FLSSM19, FLMP20, FLSSZ21a, FLSSZ21b]. These chapter show that novel and computationally effective mathematical programming formulations can be derived thanks to the bilevel interpretation of the problems. Motivated by these studies, we use bilevel programming to model the Collapsed k-Core Problem, discarding the time index.

A bilevel program is an optimization problem where one problem is nested into another

[VC94, CS07, Dem02, Cer21, KLLS21]. The formulation of a classical bilevel problem reads

$$\begin{aligned} &\min_{x\in\mathcal{X},y} F(x,y) \\ &\text{s.t. } G(x,y) \geq 0 \\ &\quad y \in \arg\min_{y'\in\mathcal{Y}}\{f(x,y') \mid g(x,y') \geq 0\} \end{aligned} \tag{P}$$

The outer optimization problem in the variables x and y is the so-called upper-level problem. The inner optimization problem in the variable y', parameterized with respect to the upper-level variables x, is the so-called lower-level problem. In formulation (P), we implicitly assume that the lower-level problem has only one optimal solution for each value of x. If this is not the case, this formulation is the one obtained with the so-called *optimistic* approach [Dem02], which consists in selecting the lower-level optimal solution corresponding to the best outcome for the upper level that minimizes it. The whole bilevel problem can be seen as a hierarchical decision process: in the upper level, a *leader* makes a decision while anticipating the optimal reaction of the lower-level decision maker, the *follower*, whose decision depends on the decision of the leader. Another way to formulate problem (P) is

$$\begin{aligned} &\min_{x\in\mathcal{X},y} F(x,y) \\ &\text{s.t. } G(x,y) \geq 0, \quad g(x,y) \geq 0 \\ &\quad f(x,y) \leq \varphi(x), \end{aligned}$$

where $\varphi(x) = \min_{y'\in\mathcal{Y}}\{f(x,y') \mid g(x,y') \geq 0\}$ is the so called *value function* of the lower level.

In the Collapsed k-Core Problem, the hierarchical structure can be described as follows. We can see the follower as an entity who is computing the collapsed k-core resulting after the decision of the leader on the b nodes to interdict from the network. So, the follower aims at identifying the subgraph of maximum cardinality, resulting from the interdiction of the b nodes selected by the leader, satisfying the property that each node of the subgraph has at least k neighbors. The leader instead aims at detecting the set of b nodes for which the cardinality of the associated subgraph is minimized, which corresponds to the set of the most critical users in the network.

It is well-known that the problem of finding the k-core of a graph, denoted as k-Core Detection Problem in the following, can be solved in polynomial time (see [BZ03]), and one can easily model the problem using binary variables. However, to the best of our knowledge, prior to the current work, no Integer Linear Programming (ILP) or Linear Programming (LP) formulation (and, thus, a polynomial approach based on LP) for the k-Core Detection Problem was known. In this work we provide a first compact LP formulation for calculating the k-core. In addition, as far as we know, no mathematical programming formulations for the Collapsed k-Core Problem have ever been investigated in the existing literature. We provide three integer programming formulations, and propose to enhance them with a combinatorial lower bound and valid inequalities. The first formulation is a compact time-expanded model that mimics the iterative node "collapsing"

process. The two other models are based on the bilevel interpretation of the problem. The first is a sparse formulation that projects out the lower-level variables and exploits a Benders-like decomposition. The second exploits the LP-based formulation of the k-Core Detection Problem, and the LP-duality. All the three formulations are implemented and computationally evaluated against a state-of-the-art bilevel solver from [FLMS17]. The obtained results demonstrate the efficiency of the proposed approaches, with the model using the LP-duality exhibiting the best performances, both in terms of computing times and gaps at termination.

3.1 Literature Review

We consider the following definition of the k-core:

Definition 1. *Given an undirected graph $G = (V, E)$, and a positive integer k, the k-core of G is the maximal induced subgraph of G in which all the nodes have degree at least k.*

The k-core may be calculated as the resulting graph obtained by iteratively deleting from G all the nodes that have degree less than k, in any order. This procedure is known as the k-core decomposition [SP16], the basic theory of which is surveyed in [MGPV20]. We point out that the original definition of the k-core given by [Sei83] requires the connectedness of the k-core. In [MB83], this original definition is used, and an algorithm, named Level Component Priority Search, is introduced for finding all the maximal connected components of a graph with degree at least k. However, in most of the recent papers in the field, the connectedness requirement is relaxed, starting from [BZ03], where an algorithm with time complexity $O(|E|)$ for core decomposition is proposed. This is why in Definition 1 we also assume that the k-core may contain multiple connected components.

The *Anchored k-Core Problem* is studied by [BKL+15]. It consists in anchoring b nodes (nodes that remain engaged no matter what their friends do) to maximize the size of the corresponding anchored k-core, i.e., the maximal subgraph in which every non-anchored node has degree at least k. It may be useful to incentivize key individuals to stay engaged within the network, preventing the cascade effect. The Anchored k-Core Problem is solvable in polynomial time for $k \leq 2$, but is NP-hard for $k > 2$ [BKL+15]. In [ZZZ+16] a greedy algorithm, called OLAK, is proposed to solve this problem, while in [LSER+20] the so-called Residual Core Maximization heuristic is introduced. Another paper focusing on the maximization of the k-core is [CFG13], where improved complexity results for the Anchored k-Core Problem are given.

From an antagonistic perspective, a natural question associated with the Anchored k-Core Problem is how to maximally collapse the engagement of the network by incentivizing the b most critical users to leave. This problem is called the *Collapsed k-Core Problem*, and was introduced in [ZZQ+17]. The aim is to find the set of b nodes the deletion of which leads to the smallest k-core (i.e., the k-core with minimum cardinality), obtained from G by iteratively removing nodes with degree strictly lower than k. In [ZZQ+17], it is showed that the Collapsed k-Core Problem is NP-hard for any $k \geq 1$, and a greedy algorithm to compute feasible solutions for the problem is proposed. In [LMS21], the parameterized complexity of the Collapsed k-Core Problem is studied

with respect to the parameters b, k, as well as another parameter, which represents the maximum allowed cardinality of the remaining k-core (i.e., it should not have more than *this number* of nodes). A further study on the k-core minimization was conducted in [ZCWL18], where the focus was on edges instead of nodes. Indeed, the aim of the problem in this case is to identify a set of b edges, so that the minimal k-core is obtained by deleting these edges from the graph. This problem is proven to be NP-hard, and a baseline greedy algorithm is proposed.

In [YLR+19], the Collapsed k-Core Problem is applied to determine the key scholars who should be engaged in various tasks in science and technology management, such as experts finding, collaborator recommendation, and so on. More recently, in [ZY20], a related problem is studied: the Collapsed (α, k)-NP-community in signed graph. Given a signed graph, which is a graph where each edge has either a "+" sign if representing a friendship or a "−" sign if representing an enemy relationship, the (α, k)-NP-community is made of the users having not less than α friends and no more than k enemies in the community. This problem aims at finding the set of nodes the removal of which from the graph will lead to the minimum cardinality of the remaining (α, k)-NP-community.

In this study, we focus on the Collapsed k-Core Problem introduced by [ZZQ+17], for which we propose three mathematical programming formulations. The first one uses variables indexed on the deletion rounds following the removal of b nodes from the graph. The other two are bilevel formulations, very similar in the basic idea, but different in the solution approaches. The lower-level problems of both the two bilevel formulations, which may be solved in polynomial time, consist in computing the k-core, i.e. model the so-called k-Core Detection Problem. To the best of our knowledge, this is the first time a mathematical programming formulation is proposed for the k-Core Detection Problem. Furthermore, this is the first work proposing mathematical programming formulations of the Collapsed k-Core Problem, and using bilevel optimization to deal with it.

3.2 Notation

Let us consider an undirected graph $G = (V, E)$, with $n = |V|$, and two positive integers k and b. Denote by $N(i)$ the set of neighbors of node i in graph G, i.e., the set of nodes $\{j \in V : \{i, j\} \in E\}$. Furthermore, denote by $\delta(G)$ the minimum degree of the graph G, which is the minimum of its nodes' degrees. For a given set of nodes $S \subseteq V$, let $G[S]$ denote the subgraph of G induced by S, and thus $\delta(G[S])$ its minimum degree. Moreover, let $G \backslash S$ denote the subgraph of G induced by $V \backslash S$, i.e., $G[V \backslash S]$. Let $C_k(G)$ be the k-core of graph G, i.e., the maximum cardinality subgraph in which every node has a degree at least k. In the following, we will refer to the number of nodes in a given k-core as its *size* or *cardinality*. Let us further define the *k-subcores* of G as the induced subgraphs of G in which all the nodes have degree at least k, but not necessarily of maximum cardinality. Finally, we say that a node v of G has coreness (or core number) k if it belongs to a k-subcore, but not to any $(k+1)$-subcore.

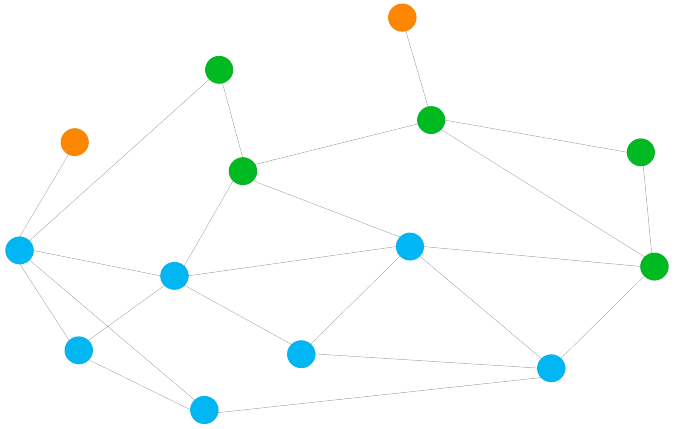


Figure 3.1: Example of coreness distribution.

In Figure 3.1, a graph is shown and each node is colored according to its core number. In particular, the two orange nodes have coreness 1, meaning that they do not belong to any k-subcore with $k \geq 2$; the five green nodes, instead, have coreness two, since 2 is the maximum value of k such that there exist a k-subcore including any of them; finally, the remaining seven cyan nodes have coreness 3, indeed they together form a 3-subcore.

3.3 Mathematical formulations for the k-Core Detection Problem

In this section we propose two formulations for the k-Core Detection Problem. The first one is an ILP model defined in the space of binary variables associated with the set of nodes. The second one is a compact, extended formulation which models the subgraph induced by the set of nodes that are outside the k-core. In this second model, which uses node and edge variables, the integrality constraints can be relaxed, and hence we obtain a LP formulation.

3.3.1 ILP Formulation

The binary variables involved in the first formulation are defined as:

$$y_i = \begin{cases} 1 & \text{if node } i \text{ belongs to the } k\text{-core} \\ 0 & \text{otherwise} \end{cases} \qquad i \in V. \tag{3.1}$$

Let

$$\mathcal{Y} = \left\{ y \in \{0,1\}^n : k y_i \leq \sum_{j \in N(i)} y_j, \, \forall \, i \in V \right\} \tag{3.2}$$

be the set of incident vectors of any subgraph in G the nodes of which have degree at least k. With each $\tilde{y} \in \mathcal{Y}$ we can associate a subset of nodes $K = \{i \in V : \tilde{y}_i = 1\}$, and with the whole set $\mathcal{Y}$ of y vectors, we can associate the set $\mathcal{K}$. For a given $K \in \mathcal{K}$, any node $i \in K$ has a degree at least k in the induced subgraph $G[K]$, i.e., $\delta(G[K]) \geq k$. Note that, in this case, by definition, $G[K]$ is the k-core of $G[K]$ itself. In Section 3.2, we defined such subgraphs $G[K]$, for each $K \in \mathcal{K}$, as *k-subcores* of G.

The k-Core Detection Problem corresponds to finding the k-core of graph G, and can be modeled as the following integer program:

$$\max_{y \in \mathcal{Y}} \sum_{i \in V} y_i. \tag{3.3}$$

Note that constraints $ky_i \leq \sum_{j \in N(i)} y_j$ (in Eq. (3.2)) ensure, when maximizing $\sum_{i \in V} y_i$, that no node with induced degree lower than k is selected in the k-core.

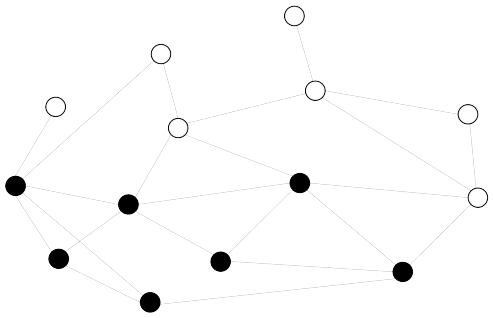


Figure 3.2: Nodes in the 3-core of graph in Figure 3.1.

In Figure 3.2, using the graph in Figure 3.1 with $k = 3$, we report in black the nodes for which the variables y_i of formulation (3.3) are 1 (in the optimal solution), i.e., the nodes belonging to the 3-core of the graph.

3.3.2 LP Formulation

In this section, we propose an alternative formulation of the k-Core Detection Problem by focusing on the subgraph of G induced by the subset of nodes which are outside the k-core. Let us define the variables u_i which identify the nodes **outside** of the k-core of the graph:

$$u_i = \begin{cases} 1 & \text{if node } i \text{ does not belong to the } k\text{-core} \\ 0 & \text{otherwise} \end{cases} \qquad i \in V. \tag{3.4}$$

Note that that variable u_i corresponds to $1 - y_i$ in formulation (3.3). Expressed in the space of u variables, the formulation (3.3) reads:

$$\max_{u} n - \sum_{i \in V} u_i \tag{3.5a}$$

$$\text{s.t.} \sum_{j \in N(i)} u_j + k - |N(i)| \leq k\, u_i \qquad \forall\, i \in V \tag{3.5b}$$

$$u \in \{0,1\}^n \tag{3.5c}$$

Indeed, we want to find the k-subcore of maximum cardinality, which means that, in the objective function (3.5a), we maximize the difference between n and the sum of u_i, that is the sum of the vertices which are outside the k-core. Constraints (3.5b) imply that u_i is 1 (i.e., node i is in not in the k-core) iff the difference between its degree and the number of neighbors not in the k-core (where this difference corresponds to the number of its neighbors in the k-core), is less than k.

Following the idea proposed in [GVPP21, Lemma 1], we can modify the right-hand side of constraints (3.5b) as:

$$\sum_{j \in N(i)} u_j + k - |N(i)| \leq \Big(\sum_{j \in N(i)} u_j + k - |N(i)| \Big) u_i. \tag{3.6}$$

If $|N(i)| - \sum_{j \in N(i)} u_j < k$ (i.e., the difference between degree of node i and the number of its neighbors not in the k-core is less than k), u_i will be set to 1. Otherwise, if $|N(i)| - \sum_{j \in N(i)} u_j \geq k$, u_i can be set either to 0 or 1, but, since we are minimizing the sum of all u_i, it will be set to 0. The resulting formulation is bilinear, and thus, possibly difficult to solve, due to the presence of the bilinear terms $u_i u_j$. We therefore linearize it through the McCormick technique, i.e., by introducing additional binary variables $x_{ij} = u_i u_j$ associated with the edges of G. Problem (3.5) can be thus reformulated as:

$$\max_{x,u} n - \sum_{i \in V} u_i \tag{3.7a}$$

$$\text{s.t.} \sum_{j \in N(i)} u_j + k - |N(i)| \leq \sum_{j \in N(i)} x_{ij} + (k - |N(i)|) u_i \qquad \forall\, i \in V \tag{3.7b}$$

$$x_{ij} \leq u_i \qquad \forall\, i \in V, j \in N(i) \tag{3.7c}$$

$$x_{ij} \leq u_j \qquad \forall\, i \in V, j \in N(i) \tag{3.7d}$$

$$x_{ij} - u_i - u_j \geq -1 \qquad \forall\, i \in V, j \in N(i) \tag{3.7e}$$

$$u \in \{0,1\}^n,\ x \in \{0,1\}^{|E|} \tag{3.7f}$$

The x variables represent the set of edges of the subgraph induced by the nodes outside of the k-core. Indeed, due to constraints (3.7c)–(3.7e), in any feasible solution of the model, two nodes i and j are connected by an edge (i.e., $x_{ij} = 1$) if and only if $u_i = u_j = 1$. Moreover, if $u_i = 0$ (i.e., the node i is in the k-core) the right-hand-side of constraints (3.7b) becomes zero because of constraints (3.7c), implying that $|N(i)| - \sum_{j \in N(i)} u_j \geq k$, which is equivalent to

$\sum_{j \in N(i)}(1 - u_j) \geq k$, guaranteeing that the number of neighbors of any node i in the k-core is at least k.

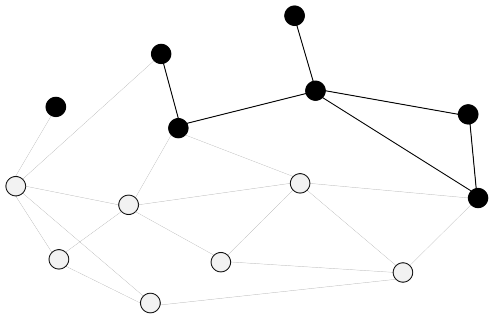


Figure 3.3: Nodes and edges in the subgraph induced by the nodes outside of the 3-core of graph in Figure 3.1.

In Figure 3.3, using the graph in Figure 3.1 with $k = 3$, we report in black the nodes i for which $u_i = 1$ and the edges $\{i, j\}$ for which $x_{ij} = 1$ in the optimal solution of formulation (3.5), i.e., the nodes and the edges of the subgraph induced by the nodes outside of the 3-core.
We now claim that the optimal solution of the continuous relaxation of the model (3.7) (i.e., the formulation obtained by replacing constraints (3.7f) with $u \in [0,1]^n,\ x \in [0,1]^{|E|}$) is integer.

Theorem 1. *Any optimal solution* (x^*, u^*) *of the continuous relaxation of formulation* (3.7) *is integer.*

Proof. Any optimal solution of the continuous relaxation of formulation (3.7) is feasible, i.e., it satisfies the following constraints

$$\sum_{j \in N(i)} u_j^* + (k - |N(i)|) \leq \sum_{j \in N(i)} x_{ij}^* + (k - |N(i)|)u_i^*,\ \forall\, i \in V. \tag{3.8}$$

For a certain node i, let $N_1(i)$ be the set of nodes $j \in N(i)$ such that $u_j^* = 1$. For the nodes in the set $N_1(i)$, $x_{ij}^* = u_i^*$ because of constraints (3.7c) and (3.7e). We can thus write

$$\sum_{j \in N(i)} u_j^* = \sum_{j \in N_1(i)} u_j^* + \sum_{j \in N(i) \setminus N_1(i)} u_j^* = |N_1(i)| + \sum_{j \in N(i) \setminus N_1(i)} u_j^*,$$

and

$$\sum_{j \in N(i)} x_{ij}^* = \sum_{j \in N_1(i)} u_i^* + \sum_{j \in N(i) \setminus N_1(i)} x_{ij}^* = |N_1(i)|u_i^* + \sum_{j \in N(i) \setminus N_1(i)} x_{ij}^* \leq |N_1(i)|u_i^* + \sum_{j \in N(i) \setminus N_1(i)} u_j^*,$$

the last inequality coming from the McCormick constraint (3.7d). We can thus reformulate (3.8) as:

$$|N_1(i)| + \sum_{j \in N(i) \setminus N_1(i)} u_j^* + (k - |N(i)|) \leq |N_1(i)|u_i^* + \sum_{j \in N(i) \setminus N_1(i)} u_j^* + (k - |N(i)|)u_i^*, \tag{3.9}$$

which can be reduced to

$$\Big[|N_1(i)| + (k - |N(i)|)\Big](1 - u_i^*) \leq 0. \tag{3.10}$$

If $|N_1(i)| + (k - |N(i)|) > 0$, then $u_i^* = 1$. Otherwise, if $|N_1(i)| + (k - |N(i)|) \leq 0$ u_i^* can have any value in $[0, 1]$, but since we are minimizing, and reducing the value of u_i does not impact on the feasibility of McCormick constraints, it will be set to 0. Thus, in the optimal solution of the continuous relaxation of (3.7), each u_i has value equal to either 0 or 1, thus the solution is integer. □

Thus, the linear relaxation of problem (3.7) provides the optimal solution of formulation (3.5).

We can further notice that constraints (3.7e), and $x \leq 1$, can be dropped. Indeed, they are redundant and guaranteed by the remaining constraints. The resulting relaxed problem is

$$\max_{x,u} n - \sum_{i \in V} u_i \tag{3.11a}$$

$$\text{s.t.} \sum_{j \in N(i)} u_j + k - |N(i)| \leq \sum_{j \in N(i)} x_{ij} + (k - |N(i)|)u_i \qquad \forall\, i \in V \tag{3.11b}$$

$$x_{ij} \leq u_i \qquad \forall\, i \in V, j \in N(i) \tag{3.11c}$$

$$x_{ij} \leq u_j \qquad \forall\, i \in V, j \in N(i) \tag{3.11d}$$

$$u \in [0, 1]^n,\ x \in {\mathbb{R}_+}^{|E|} \tag{3.11e}$$

which is linear in the variables x and u.

3.4 Mathematical formulations for the Collapsed k-Core Problem

Leveraging on the formulations presented above for the k-Core Detection Problem, in this section, we propose three models for the Collapsed k-Core Problem, which is defined as follows.

Definition 2. *Given an undirected graph $G = (V, E)$ and two positive integers k and b, the Collapsed k-Core Problem consists of finding a subset $W^* \subset V$ of b nodes, the removal of which minimizes the size of the resulting k-core (i.e., for which $|C_k(G \backslash W^*)|$ is minimum).*

In the following, we will refer to these b nodes as *collapsers*.

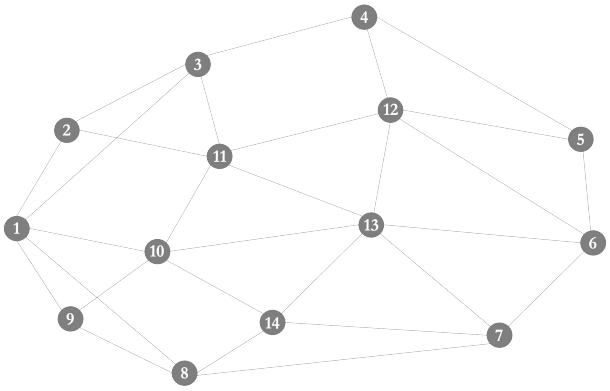
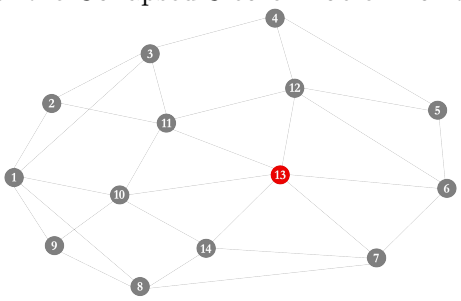
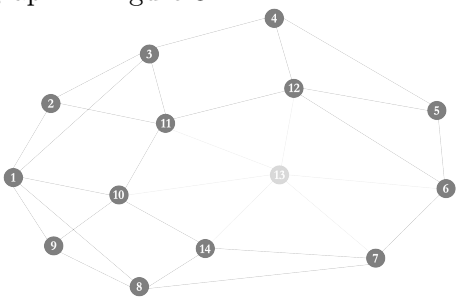


Figure 3.4: Example graph for the Collapsed k-Core.

In Figures 3.4, and 3.5 some of the introduced concepts are visualized through examples. The graph G with 14 nodes in Figure 3.4 is a 3-core. Indeed, every node has degree at least 3. The subgraph induced by the set $S = \{6, 7, 10, 11, 12, 13, 14\}$ is a 3-subcore, i.e., S is a set of nodes such that the degree of $G[S]$ is three, but its cardinality is not maximum. Assume that we want to determine the Collapsed 3-core Problem on this graph with budget $b = 1$, i.e., we can remove one node only. Two possible feasible solutions of this Collapsed 3-core Problem are represented in Figure 3.5 and Figure 3.6. The first one in Figure 3.5, which consists in removing node 13 from the graph, leads to a 3-core of cardinality 13 (no node follows node 13, because no node has less than 3 neighbors in the remaining graph). The second one in Figure 3.6, which consists in removing node 1 from the graph, leads to a better solution, since the obtained 3-core after the cascade effect following node 1 removal consists in 7 nodes. In fact, this is the optimal solution of the Collapsed 3-core Problem for the graph in Figure 3.4.

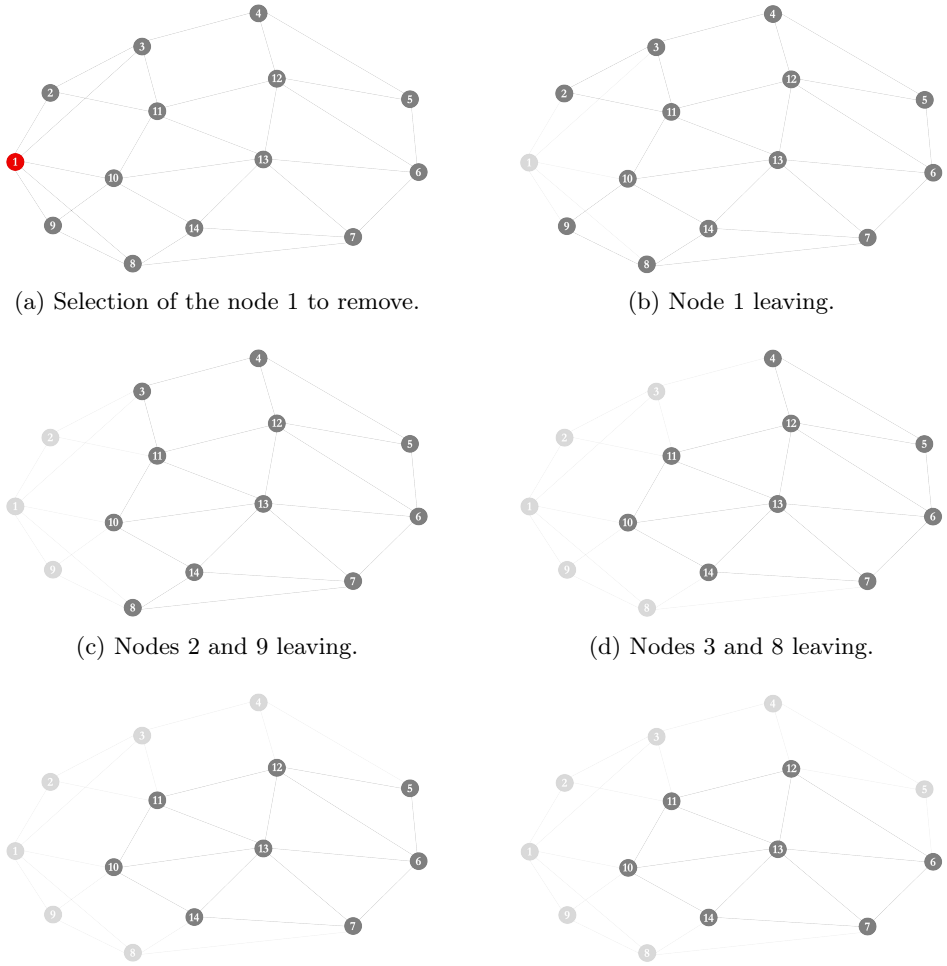


(a) Selection of the node 13 to remove.



(b) Node 13 leaving.

Figure 3.5: Feasible and suboptimal solutiom.



(a) Selection of the node 1 to remove.

(b) Node 1 leaving.

(c) Nodes 2 and 9 leaving.

(d) Nodes 3 and 8 leaving.

(e) Node 4 leaving.

(f) Node 5 leaving.

Figure 3.6: Optimal solution for the input graph in Figure 3.5.

In the rest of this section, we propose three formulations for the problem: the first one relies on the iterative process used to determine the collapsed k-core, after the removal of the b nodes; the other two formulations are bilevel formulations, considering two agents, a leader who selects the nodes to remove and a follower who computes the resulting k-core, solving either formulation (3.3) or (3.11). The bilevel structure of the problem comes from the fact that the k-core is defined as the induced subgraph with all nodes having degree at least k of maximum size. Given that, the aim of the problem is to find the set of b nodes to remove, s.t. the maximal induced k-subcore is of minimum cardinality.

3.4.1 Time-Dependent Formulation

In this section, we describe a "natural" ILP formulation for the Collapsed k-Core Problem. This formulation models the so-called cascade effect deriving from the removal of b nodes, which are removed at time 0. At each deletion round (corresponding to a time instant t), all the nodes the degree of which becomes less than k are removed from the graph. Assuming that the problem instance is feasible, i.e., that the considered graph has at least b nodes, the deletion rounds are at most $T = n - b$.

We introduce the binary variables a_i^t, defined for each node $i \in V$ and time $t = 0, \ldots, T$, such that:

$$a_i^t = \begin{cases} 1 & \text{if node } i \text{ belongs to the induced subgraph at time } t \\ 0 & \text{if node } i \text{ does not belong to the induced subgraph at time } t \end{cases}$$

In the example shown in Figure 3.6, e.g., $a_1^0 = 0$ and $a_i^0 = 1$ for $i \in V \backslash \{1\}$, while, as shown in Figure 3.6f, $a_i^T = 1$ for $i = \{6, 7, 10, 11, 12, 13, 14\}$ and $a_i^T = 0$ for $i = \{1, 2, 3, 4, 5, 8, 9\}$, where $T = 4$.

The Collapsed k-Core Problem can be formulated as:

$$\min_a \sum_{i \in V} a_i^T \tag{3.12a}$$

$$\text{s.t.} \sum_{i \in V} a_i^0 = n - b \tag{3.12b}$$

$$a_i^t \leq a_i^{t-1} \qquad \forall\, i \in V,\ t = 1, \ldots, T \tag{3.12c}$$

$$\sum_{j \in N(i)} a_j^{t-1} - k + 1 \leq d(i) a_i^t + d(i)(1 - a_i^0) \qquad \forall\, i \in V,\ t = 1, \ldots, T \tag{3.12d}$$

$$a_i^t \in \{0, 1\} \qquad \forall\, i \in V,\ t = 0, \ldots, T \tag{3.12e}$$

where $d(i) = |N(i)| - k + 1$. The objective function minimizes the number of nodes remaining in the last time instant T. The first constraint (3.12b) imposes that at the beginning (first time instant) exactly b nodes are removed. Constraints (3.12c) state that if node i is not in the graph at time $t - 1$, it cannot be in the graph at time t. Finally, constraints (3.12d) ensure that, if the node is not removed at time 0, it must stay in the graph at time t iff more than $k - 1$ of its neighbors "survived" at time $t - 1$. Constraint (3.12d), for a given $i \in V$, and a given $t \in \{1, \ldots, T\}$, imposes that

- if the node is not a collapser, i.e., it is not removed at time 0, and thus $a_i^0 = 1$
- and $\sum_{j \in N(i)} a_j^{t-1} \geq k$, i.e., $\sum_{j \in N(i)} a_j^{t-1} - k + 1 \geq 1$ (with $\sum_{j \in N(i)} a_j^{t-1} \leq |N(i)|$)

then a_i^t is set to 1. The term $(|N(i)| - k + 1)(1 - a_i^0)$ is thus needed to guarantee that, for the collapsers, the constraints (3.12d) are still satisfied. For these nodes indeed a_i^0 and a_i^t will be 0.

The obtained ILP formulation is polynomial in size and can be solved using existing state-of-the-art solvers. It is indeed a compact formulation, which is the main advantage of this first natural approach. However, because of time index t, the number of variables is large, thus requiring a long computational time to be solved. For this reason, we present in the following sections two alternative formulations for the Collapsed k-Core Problem.

3.4.2 A first bilevel formulation

In this section, we formulate the Collapsed k-Core Problem using bilevel programming where the leader aims at *minimizing* the cardinality of the k-core obtained by removing exactly b nodes. The follower instead aims at detecting the k-core obtained after the removal of the b nodes chosen by the leader, which corresponds to finding the *maximal* induced subgraph where all the nodes have degree at least k. The follower's problem is modeled as in formulation (3.3), with additional linking constraints imposing that the k-core is computed in the graph resulting after the removal of b nodes by the leader.

We consider the upper-level binary variables w_i and the lower-level binary variables y_i (already defined in formulation (3.1)), both defined for each node $i \in V$:

$$w_i = \begin{cases} 1 & \text{if node } i \text{ is removed by the leader} \\ 0 & \text{otherwise} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if node } i \text{ is in the collapsed } k\text{-core of the follower} \\ 0 & \text{otherwise} \end{cases}$$

Variable w_i is 1 iff node i is a collapser. We remark that it corresponds to $1 - a_i^0$ in the time-dependent formulation (3.12), presented in Subsection 3.4.1. Variable y_i, instead, is 1 iff node i is in the resulting k-core, thus corresponds to a_i^T in the time-dependent formulation (3.12).
In the example shown in Figure 3.6, e.g., $w_1 = 1$ and $w_i = 0$ for $i \in V \backslash \{1\}$, while, as shown in Figure 3.6f, $y_i = 1$ for $i = \{6, 7, 10, 11, 12, 13, 14\}$ and $y_i = 0$ for $i = \{1, 2, 3, 4, 5, 8, 9\}$.
The set of all possible leader's policies $\mathcal{W}$ (removing exactly b nodes) is

$$\mathcal{W} = \left\{ w \in \{0,1\}^n : \sum_{i \in V} w_i = b \right\}. \tag{3.13a}$$

Let Ω denote the set of all subsets $W \subseteq V$ such that $|W| = b$. There is a one-to-one correspondence between each element $W \in \Omega$ and its incidence vector $w \in \mathcal{W}$. The set of all possible nodes subsets inducing k-subcores of G is $\mathcal{K}$, defined in Section 3.3.1. If no node is interdicted by the leader, the problem of the follower corresponds to finding the k-core in the original graph, defined in Section 3.3.1. The resulting Collapsed k-Core Problem can be formulated as the following bilevel problem:

$$\min_{w \in \mathcal{W}} \max_{y \in \mathcal{Y}} \left\{ \sum_{i \in V} y_i \ : \ y_i \leq 1 - w_i, \ \forall\, i \in V \right\}. \tag{3.14}$$

Constraints $y_i \leq 1 - w_i$ exclude from the k-core the collapsers. Problem (3.14) does not consider the optimal solutions of the follower's problem, but only its optimal objective function value. Thus, we do not need to distinguish between optimistic and pessimistic concepts.

3.4.3 Sparse Formulation

Formulation (3.14) exhibits the structure of so-called *interdiction* problems, which is a well-known class of bilevel optimization problems (see [KLLS21, SS20] for recent surveys on interdiction problems). This structure allows us to apply a Benders-like reformulation technique in which we project out the lower-level variables and introduce an auxiliary integer variable z to represent the objective value of the lower-level problem. We refer to this formulation as *sparse*, because it is given in the natural space of w variables (required to describe the removed set of nodes) and a single auxiliary variable. We start by reformulating the problem (3.14) as follows:

$$\min_{z \in \mathbb{Z}, w \in \{0,1\}^n} z \tag{3.15a}$$

$$\text{s.t. } z \geq \max_{y \in \mathcal{Y}} \left\{ \sum_{i \in V} y_i \,:\, y_i \leq 1 - w_i, \forall\, i \in V \right\} \tag{3.15b}$$

$$\sum_{i \in V} w_i = b. \tag{3.15c}$$

Following the ideas from, e.g., [Woo11, FLMS19, LLM+22], we can then reformulate (3.15b), given sufficiently large M_i for all i, as:

$$z \geq \max_{y \in \mathcal{Y}} \left\{ \sum_{i \in V} y_i - \sum_{i \in V} M_i y_i w_i \right\} \tag{3.16}$$

which is equivalent to the following Benders-like constraints:

$$z \geq \sum_{i \in V} \hat{y}_i - \sum_{i \in V} M_i w_i \hat{y}_i \quad \forall\, \hat{y} \in \mathcal{Y}. \tag{3.17}$$

In terms of sets $K \in \mathcal{K}$ inducing the k-subcores of G, we can rewrite Ineqs. (3.17) as

$$z \geq |K| - \sum_{i \in K} M_i w_i \quad \forall\, K \in \mathcal{K}. \tag{3.18}$$

The value of M_i needs to be set in such a way that, for a given K, in case node i (possibly together with some other nodes from K) is interdicted, the value $|K| - M_i$ gives the lower bound on the size of the k-core of $G[K]$. Since, in the extreme case, the interdiction can lead to an empty k-core, we set $M_i = |K|$, resulting into

$$z \geq |K| \left[1 - \sum_{i \in K} w_i \right] \quad \forall\, K \in \mathcal{K}. \tag{3.19}$$

Alternatively, since the upper-level decisions are binary, we can reformulate constraint (3.15b) using the following $\binom{n}{b}$ many no-good-cuts:

$$z \geq |C_k(G\backslash W)| \left[\sum_{i \in W} w_i - b + 1 \right], \qquad \forall\, W \in \Omega. \tag{3.20}$$

Indeed, for any given W, the cardinality of the k-core $C_k(G\backslash W)$, when the nodes from W are removed by the leader, provides a valid lower bound on z. If at least one of the nodes in W is not a collapser, the related constraint of type (3.20) turns out to be redundant, since the right hand side becomes less than or equal to 0.

In Subsection 3.6.2, we will present a separation procedure to solve the single-level reformulation of (3.15) given as

$$\min_{z \in \mathbb{Z}, w \in \mathcal{W}} z \tag{3.21a}$$

$$\text{s.t. } (3.19),\ (3.20) \tag{3.21b}$$

which has exponentially many constraints.

3.4.4 A second bilevel formulation

In this section, we propose an alternative bilevel formulation by considering lower-level variables u_i defined in equation (3.4), complementary with respect to variables y_i, and formulating the lower-level problem as in formulation (3.11).

We recall that u_i represent the lower-level variables identifying the nodes not belonging to the collapsed k-core of the graph:

$$u_i = \begin{cases} 1 & \text{if node } i \text{ does not belong to the collapsed } k\text{-core of the follower} \\ 0 & \text{otherwise} \end{cases}$$

We remark that variable u_i corresponds to $1 - y_i$ in the former bilevel formulation. Indeed, in the example shown in Figure 3.6, $u_i = 1$ for $i = \{1, 2, 3, 4, 5, 8, 9\}$, and $u_i = 0$ for $i = \{6, 7, 10, 11, 12, 13, 14\}$.

The problem of detecting the k-core of the graph, given a leader's decision $w \in \mathcal{W}$, can be modelled as the problem of determining the set of nodes outside of the k-core:

$$\Psi(w) := \min_{u} \sum_{i \in V} u_i \tag{3.22a}$$

$$\text{s.t. } \sum_{j \in N(i)} u_j + k - |N(i)| \leq k\, u_i \qquad \forall\, i \in V \tag{3.22b}$$

$$w_i \leq u_i \qquad \forall\, i \in V \tag{3.22c}$$

$$u \in \{0, 1\}^n \tag{3.22d}$$

This formulation differs from formulation (3.5) only for constraints (3.22c), which state that a node i cannot be in the k-core if it is interdicted/removed.

The corresponding bilevel formulation of the Collapsed k-Core Problem is:

$$\min_{v\in\mathbb{Z},w\in\{0,1\}^n} n - v \tag{3.23a}$$

$$\text{s.t. } v \leq \Psi(w) \tag{3.23b}$$

$$\sum_{i\in V} w_i = b. \tag{3.23c}$$

The objective function expresses the fact that we want to find the minimal collapsed k-core, computed by solving $\Psi(w)$.

In Section 3.3.2, we proved that problem $\Psi(w)$ can be reformulated as the following LP formulation:

$$\min_{u} \sum_{i\in V} u_i \tag{3.24a}$$

$$\text{s.t. } \sum_{j\in N(i)} u_j + k - |N(i)| \leq \sum_{j\in N(i)} x_{ij} + (k - |N(i)|)u_i \qquad \forall\, i \in V \tag{3.24b}$$

$$x_{ij} \leq u_i \qquad \forall\, i \in V, j \in N(i) \tag{3.24c}$$

$$x_{ij} \leq u_j \qquad \forall\, i \in V, j \in N(i) \tag{3.24d}$$

$$u_i \geq w_i \qquad \forall\, i \in V \tag{3.24e}$$

$$u \in [0,1]^n,\ x \in {\mathbb{R}_+}^{|E|} \tag{3.24f}$$

The addition of constraints (3.24e), indeed, has no impact on the proof of Theorem 1. Since formulation (3.24) is linear in the variables x, and u, we can replace it by its dual, as detailed in the following subsection.

3.4.5 Compact nonlinear formulation

An approach to deal with the bilevel formulation (3.23) consists in dualizing the lower level continuous formulation (3.24). Let us define the following dual variables for all $i \in V$:

- α_i associated with the constraints (3.24b),
- $\beta_{ij}, \forall\, j \in N(i)$ associated with the constraints (3.24c),
- $\gamma_{ij}, \forall\, j \in N(i)$ associated with the constraints (3.24d),
- λ_i associated with the constraints (3.24e),
- τ_i associated with the constraints $u_i \leq 1$.

The dual of the the lower-level problem (3.24) is:

$$\max_{\alpha,\beta,\gamma,\lambda,\tau} \sum_{i\in V} [(k - |N(i)|)\alpha_i + w_i\lambda_i - \tau_i] \tag{3.25a}$$

$$(k - |N(i)|)\alpha_i + \lambda_i - \tau_i + \sum_{j\in N(i)} (-\alpha_j + \beta_{ij} + \gamma_{ji}) \leq 1 \quad \forall\, i \in V \tag{3.25b}$$

$$\alpha_i - \beta_{ij} - \gamma_{ij} \leq 0 \qquad \forall\, i \in V, j \in N(i) \tag{3.25c}$$

$$\alpha_i, \lambda_i, \tau_i \geq 0 \qquad \forall\, i \in V \tag{3.25d}$$

$$\beta_{ij}, \gamma_{ij} \geq 0 \qquad \forall\, i \in V, j \in N(i) \tag{3.25e}$$

For any value of w, problem (3.24) *(i)* admits at least one feasible solution, *(ii)* is bounded because both variables x and u are bounded. Thus, strong duality holds between problem (3.24) (the LP relaxation of Ψ) and its dual (3.25). Given what we discussed before, in (3.23), we can replace $\Psi(w)$ by its linear relaxation (3.24) since their optimal values are the same as proved in Theorem 1, and then replace problem (3.24) by (3.25), since their optimal values are the same by strong duality. We can further drop the maximum operator, obtaining the following single-level formulation:

$$\min_{v,w,\alpha,\beta,\gamma,\lambda,\tau} n - v \tag{3.26a}$$

$$\text{s.t. } v \leq \sum_{i\in V} [(k - |N(i)|)\alpha_i + w_i\lambda_i - \tau_i] \tag{3.26b}$$

$$\sum_{i\in V} w_i = b \tag{3.26c}$$

$$(k - |N(i)|)\alpha_i + \lambda_i - \tau_i + \sum_{j\in N(i)} (-\alpha_j + \beta_{ij} + \gamma_{ji}) \leq 1 \quad \forall\, i \in V \tag{3.26d}$$

$$\alpha_i - \beta_{ij} - \gamma_{ij} \leq 0 \qquad \forall\, i \in V, j \in N(i) \tag{3.26e}$$

$$\alpha_i, \lambda_i, \tau_i \geq 0 \qquad \forall\, i \in V \tag{3.26f}$$

$$\beta_{ij}, \gamma_{ij} \geq 0 \qquad \forall\, i \in V, j \in N(i) \tag{3.26g}$$

$$v \in \mathbb{Z},\ w \in \{0,1\}^n. \tag{3.26h}$$

This single-level formulation is a Mixed-Integer Nonlinear Programming (MINLP) problem, that has bilinear terms in (3.26d), given by $\sum_{i\in V} w_i\lambda_i$. One could linearize these bilinear terms using again McCormick reformulation, and/or applying other specialized techniques. However, most of the these state-of-the-art techniques are integrated in modern MINLP solvers, thus we decided to hand over the compact model (3.26) as it is to the solver used in the experiments (see Section 3.6.3 for details).

3.5 Valid inequalities

In this section, we describe different classes of valid inequalities which are used to strengthen the formulations presented above. We point out that an initial *pre-processing* procedure is applied to G which consists of removing all nodes not belonging to its k-core.

3.5.1 Dominance and symmetry breaking inequalities

For any node $u \in V$, we can compute the k-core of $G \setminus \{u\}$ and define J_u as the set of nodes, including u itself, which leave the graph when node u is removed (say, the *followers* of u, not to be confused with the follower agent solving the lower level):

$$J_u = \{v \in V : v \notin C_k(G \backslash \{u\})\}.$$

In the example graph in Figure 3.4, as shown in Figures 3.5 and 3.6, with $k = 3$, $J_{13} = \{13\}$, while $J_1 = \{1, 2, 3, 4, 5, 8, 9\}$. Using these sets, defined for each node in the graph, we can add dominance inequalities to our formulations. In the same example as before, $J_3 = \{2, 3, 4, 5\}$ and we can observe that $J_3 \subset J_1$, meaning that every node that leave the network when node 3 is removed, would also leave the network when node 1 is removed. In general, if $J_i \subset J_j$, i.e., the set of followers of node i is strictly contained in the set of followers of node j, node j should be removed first. This can be imposed adding to the time-dependent formulation (3.12) the following inequalities:

$$a_j^0 \leq a_i^0 \qquad \forall\, i, j \in V : J_i \subset J_j \tag{3.27}$$

which corresponds to adding the following inequalities to the upper level of the bilevel formulations:

$$w_j \geq w_i \qquad \forall\, i, j \in V : J_i \subset J_j. \tag{3.28}$$

An additional family of valid inequalities is related to breaking symmetries among nodes which have the same set of followers. Let $S = \{i_1, \ldots, i_{|S|}\}$ be an inclusion-wise maximal subset of nodes such that $J_{i_r} = J_{i_s}$ for all $i_r, i_s \in S$, i.e., S contains all the nodes of the graph having a given set of followers, and there exist no superset of S the nodes of which have the same set of followers. For example, in the graph in Figure 3.4, when $k = 3$, nodes 4 and 5 have the same set of followers which is $\{4, 5\}$ (if 4 leaves the network, 5 leaves it too and vice versa), thus a possible set S is $\{4, 5\}$. Let the indices i_r, with $r \in \{1, \ldots, |S|\}$, be given in increasing order. Then we can break the symmetries by imposing that the node with the lowest index is removed first, i.e.,

$$a_{i_1}^0 \leq \cdots \leq a_{i_{|S|}}^0 \qquad \forall\, S \subset V : J_{i_r} = J_{i_s}, \forall\, i_r, i_s \in S \tag{3.29}$$

for the time-dependent formulation (3.12), and

$$w_{i_1} \geq \cdots \geq w_{i_{|S|}} \qquad \forall\, S \subset V : J_{i_k} = J_{i_s}, \forall\, i_r, i_s \in S \tag{3.30}$$

for the upper level of formulations (3.15) and (3.23) (as well as its single-level reformulations (3.21) and (3.26)).

3.5.2 Valid inequalities to consider the cascade effect

We present in this section a family of valid inequalities, related to the nodes which leave the network as a consequence of a single node or a set of nodes leaving. Such inequalities can be added to both the time-dependent formulation (3.12) and the leader's problem of the two bilevel formulations.

For any node $u \in V$, we can compute the k-core of $G \setminus \{u\}$ and the set of followers J_u. Given that by removing u, all nodes in J_u will be removed as well, we can add a valid inequality stating that at most one node should be removed from J_u, that is:

$$\sum_{j \in J_u} a_j^0 \geq |J_u| - 1 \qquad \forall\, u \in V, \tag{3.31}$$

for the time-dependent formulation, and

$$\sum_{j \in J_u} w_j \leq 1 \qquad \forall\, u \in V, \tag{3.32}$$

for the upper level of the bilevel formulations. We assume that the removal of less than b nodes (together with the related followers) is not enough to empty the network. Under this assumption, Ineqs. (3.12b), (3.15c), (3.23c), requiring that exactly b nodes are removed, as well as Ineqs. (3.19) and (3.20), remain still valid when introducing constraints (3.31) and (3.32).

Valid inequalities (3.31) and (3.32) can be generalized to the case in which more than a single node is removed. Let $S \subseteq V$, with $|S| < b$, be such set of nodes. Assume

$$|C_k(G \backslash S)| \geq b - |S|$$

holds, i.e., in the remaining k-core there are enough nodes to remove according to the budget left. The set J_S of followers of S (including S) can be defined as follows

$$J_S = \{j \in V : j \notin C_k(G \backslash S)\}$$

and the inequality (3.31) is generalized into:

$$\sum_{j \in J_S} a_j^0 \geq |J_S| - |S| \qquad \forall\, S \subseteq V : |S| < b, \tag{3.33}$$

while the inequality (3.32) into:

$$\sum_{j \in J_S} w_j \leq |S| \qquad \forall\, S \subseteq V : |S| < b. \tag{3.34}$$

The number of constraints (3.31) and (3.32) is equal to the number of nodes in the graph. For each node $u \in V$, the set J_u, i.e., the followers of u, can be easily obtained by computing the k-core of $G \setminus \{u\}$ and the related constraint can be added to the model. On the other hand, the number of constraints (3.33) and (3.34) is $\binom{n}{b-1}$. Thus a separation routine is required.

3.5.3 Lower bound on the solution value

Let $\{L_i\}_{i\in\{k,\dots,\ell\}}$ be a series of layers, each one containing the nodes of G with coreness (a node v of G has coreness (or core number) k if it belongs to a k-subcore, but not to any $(k+1)$-subcore) equal to i, with i being at least k and at most ℓ, where ℓ is the maximum coreness of any node in the graph. For instance, in the graph in Figure 3.1, with $k = 1$, we have $\ell = 3$, L_1 contains the cyan nodes, L_2 contains the green nodes, while L_3 contains the orange nodes.

Let us consider the h-core $\bigcup_{i\in\{h,\dots,\ell\}} L_i$, where $h = k + b$. Even by removing any subset of b nodes from such h-core, the remaining nodes still constitute a k-core. The size of the remaining k-core is a valid lower bound to the solution value of the Collapsed k-Core Problem. Let us denote as $m := \left|\bigcup_{i\in\{h,\dots,\ell\}} L_i\right| - b$ this lower bound. This means that we can restrict the set $\mathcal{Y}$ of the incident vectors of all the k-subcores of the graph (over which we optimize the lower-level problem) as follows:

$$\tilde{\mathcal{Y}} = \left\{ y \in \{0,1\}^n \in \mathcal{Y} : \sum_{j\in\bigcup_{i\in\{h,\dots,\ell\}} L_i} (1 - y_j) \leq b \right\}. \tag{3.35}$$

Indeed the number of nodes which are not in the feasible k-cores belonging to the layers $\bigcup_{i\in\{h,\dots,\ell\}} L_i$ will not be greater than the budget b.

This corresponds to adding the following constraint to the time-dependent model (3.12)

$$\sum_{i\in V} a_i^T \geq m. \tag{3.36}$$

Furthermore, a tighter upper bound on the number of deletion rounds can be defined as $T = n - b - m$.

Similarly, for the bilevel formulations, constraint

$$z \geq m, \tag{3.37}$$

can be added to the upper level of formulation (3.15) and

$$v \leq n - m, \tag{3.38}$$

to (3.23), as well as to (3.26).

According to the defined lower bound, in a similar fashion as it is done in stochastic integer programming (see, e.g. [LL93]), we can also tighten the constraints of the sparse formulation (3.21), presented in Subsec. 3.4.3. Inequalities (3.19) can be restated as follows:

$$z \geq m + (|K| - m)\left[1 - \sum_{i\in K} w_i\right] \qquad \forall\, K \in \mathcal{K} : |K| > m \tag{3.39}$$

and inequalities (3.20) as follows:

$$z \geq m + (|C_k(G\backslash W)| - m)\left[\sum_{i\in W} w_i - b + 1\right] \qquad \forall\, W \in \Omega. \tag{3.40}$$

3.5.4 Valid inequalities derived from k-subcores

For a given $K \in \mathcal{K}$, assume that $\delta(G[K]) \geq k+1$ (the degree of nodes in the subgraph of G induced by K is at least $k+1$). Assume we are given an interdiction policy $\tilde{W} \subset \Omega$ such that at most one of the nodes in K is interdicted, then we have that $|C_k(G \backslash \tilde{W})| \geq |K| - 1$. Thus we can impose:

$$z \geq m + (|K| - 1 - m) \left[1 - \sum_{i \in K} \frac{w_i}{2} \right] \qquad \forall\, K \in \mathcal{K} : \delta(G[K]) \geq k+1 \tag{3.41}$$

in the upper-level of formulation (3.15).

This can be easily generalized to the case in which $\delta(G[K]) = h \geq k+1$ as follows:

$$z \geq m + (|K| - h + k - m) \left[1 - \sum_{i \in K} \frac{w_i}{h - k + 1} \right] \quad \forall\, K \in \mathcal{K} : \delta(G[K]) = h \geq k+1. \tag{3.42}$$

3.6 Separation procedures

The inequalities (3.27), (3.28), (3.29), and (3.30) introduced in Section 3.5.1, as well as the inequalities (3.31) and (3.32) modeling the cascade effect following the leaving of a single node, introduced in Section 3.5.2, and the ones related to the combinatorial lower bound m, i.e., (3.36), (3.37) and (3.38), introduced in Section 3.5.3, are added to the corresponding models during the initialization phase as they are in polynomial number. Specifically, we add the following $|V|+1$ inequalities: the $|V|$ inequalities (3.31), or (3.32), and the inequality (3.36), or (3.37), or (3.38) (just one inequality for each model). As regards inequalities (3.27), (3.28), (3.29), and (3.30), we add them from the beginning following an heuristic procedure here described. After computing the set of followers J_j for each node $j \in V$, a dominance inequality of type (3.27) or (3.28) is added to the corresponding formulation for each node $i \in J_j$ such that $J_i \subset J_j$. Similarly, a partitioning $\mathcal{P}$ of the set of nodes V into at most $|V|$ disjoint subsets is constructed, by iteratively assigning each node $i \in V$ to the subset which contains nodes having exactly the same followers of node i. Formally, for any S in $\mathcal{P}$, $J_i = J_j, \forall\, i,j \in S$ and $J_i \neq J_j, \forall\, i \in S, \forall\, j \notin S$. Hence, a symmetry breaking inequality of type (3.29) or (3.30) is added to the appropriate formulation for each set S of the partition $\mathcal{P}$.

Instead, the other valid inequalities introduced in Section 3.5 need a procedure to be separated. In this section, we first present the separation procedure associated with compact formulations (3.12) and (3.26) and used to separate valid inequalities (3.33) and (3.34). We then describe the separation procedures for the non-compact formulation (3.21) used to separate constraints constraints (3.39), (3.40) and inequalities (3.34), and (3.42). We note that separation is made on integer solutions only.

3.6.1 Separation procedures for the compact formulations

A heuristic procedure for detecting violated inequalities (3.34) added to formulation (3.26) is here described.

1. Consider an interdiction policy $\bar{w}$ of the leader and let $\bar{W}$ denote the related set of collapsers;
2. For each node $j \in \bar{W}$ do the following:
   (a) Compute the k-core of $G \backslash \bar{W} \cup \{j\}$, i.e. $C_k(G \backslash \bar{W} \cup \{j\})$
   (b) If $j \notin C_k(G \backslash \bar{W} \cup \{j\})$, then $j \in J_{\bar{W} \setminus \{j\}}$ and add a violated inequality of type (3.34) with $S = \bar{W} \setminus \{j\}$ to formulation (3.26).

Given a set $\bar{W}$ of collapsers, this routine is able to identify violated constraints of type (3.34) where S is given by the set of collapsers $\bar{W}$ excluding exactly one of them. An analogous separation procedure is used to separate constraints (3.33) for formulation (3.12), where we consider variables $\bar{a}^0$ and the related set $\bar{A}$ of collapsers instead of $\bar{w}$ and set $\bar{W}$ used in the procedure above.

3.6.2 Separation procedures for the single-level formulation (3.21)

In the following, we explain how to heuristically separate constraints (3.34), (3.39), (3.40) and (3.42). The procedure to separate (3.34) is the same as the one described above. We repeat it here for the ease of reading.

We start initializing the relaxation of problem (3.21), obtained by dropping constraints (3.19) and (3.20). Then, every time a feasible integer solution $\hat{w}$ of such relaxation is found, we compute the corresponding $C_k(G \backslash \hat{W})$ and:

1. For each node $j \in \hat{W}$ do the following:
   (a) Compute the k-core of $G \backslash \hat{W} \cup \{j\}$, i.e. $C_k(G \backslash \hat{W} \cup \{j\})$;
   (b) If $j \notin C_k(G \backslash \hat{W} \cup \{j\})$, i.e. $j \in J_{\hat{W} \setminus \{j\}}$, add a violated inequality of type (3.34) with $S = \hat{W} \backslash \{j\}$.
2. If no violated inequality of type (3.34) is identified, go to step 3, otherwise go to step 4.
3. If $z < |C_k(G \backslash \hat{W})|$, add to the current relaxation a cut of the family (3.39) with $K = C_k(G \backslash \hat{W})$ and a cut of the family (3.40) with $W = \hat{W}$.
4. Set $U = \emptyset$, and iteratively perform the following steps:
   (a) Select a node $u \in V \setminus U$ to remove and set $U := U \cup \{u\}$;
   (b) Compute the k-core $C_k(\{G \backslash \hat{W}\} \setminus U)$;

(c) If $|C_k(\{G\backslash\hat{W}\} \setminus U)| > m$, and $z < |C_k(\{G\backslash\hat{W}\} \setminus U)|$ add a cut of the family (3.39) with $K = C_k(\{G\backslash\hat{W}\} \setminus U)$. Otherwise, go to step 5.

(d) If $|U|$ is over a given threshold, go to step 5.

5. For $h \in \{k+1, \ldots, \ell\}$:

   (a) consider the set $K = \bigcup_{h \leq i \leq \ell} L_i$ of nodes of $C_k(G\backslash\hat{W})$ having coreness at least h.

   (b) If $|K| \geq m$, then add cut (3.42).

At step 1 of the above presented procedure, we check if the collapsers are all *really useful*. Indeed, we verify whether each $j \in \hat{W}$ is a follower of the other nodes in $\hat{W}$; if this is the case, removing j is not useful for the leader: it will anyway disappear as follower of the other collapsers.

At step 2, we verify if it is needed to perform step 3. Indeed, if at least one of the inequalities (3.34) has been added to the relaxation, there is no need to cut off the current solution by means of (3.39) and (3.40), being this solution already excluded by adding constraints of type (3.34).

At step 4, we add a certain number (at most $|U|$) of Bender's like cuts of type (3.39) with K of increasingly smaller dimension. In order to obtain diversified sets of valid inequalities, nodes in $V \setminus U$ are selected in each iteration of step 4 according to decreasing order of the number of previously added constraints in which they are involved. This heuristic selection procedure means that the more a node is *involved* in the previous steps, the less it will be considered in step 4.

At step 5, at most $\ell - k$ inequalities of type (3.42) are added. In particular, for any given $h \in \{1, \ldots, \ell\}$, the set of nodes in $C_k(G\backslash\hat{W})$ having core number at least h is computed and used as the set K in (3.42).

3.6.3 Numerical experiments

In this section, we analyse the computational performances of the following four exact approaches:

- Time-Dependent Model, corresponding to the compact ILP formulation (3.12), see Section 3.4.1
- Nonlinear Model, corresponding to the compact MINLP formulation (3.26) presented in Section 3.4.5
- Sparse Model, corresponding to the non-compact formulation (3.21) (cf. Section 3.4.3) for which we implemented a Branch&Cut (B&C) approach, and
- Bilevel Solver, corresponding to the bilevel formulation (3.15) which is solved using a general purpose intersection-cut based solver for Mixed-Integer Bilevel Linear Problems proposed in [FLMS17].

On the one hand, the two compact formulations (3.12) and (3.26) are solved using a state-of-the-art MINLP solver together with the separation procedures proposed in Section 3.6.1. On the other hand, formulation (3.21) is solved using a Branch&Cut method which iteratively builds the feasible set of the original bilevel formulation (3.15), by adding cuts of type (3.19), (3.20) as well as separating the valid inequalities proposed in Section 3.5 through the separation procedure illustrated in Section 3.6.2.

The proposed formulations were implemented in Python 3.8 and solved by using the Gurobi solver (version 9.5.2). The bilevel solver of [FLMS17] uses Cplex 12.7. The separation procedures presented in the paper are implemented within lazy callbacks, with the threshold on $|U|$ used in step 4d of separation procedure presented in Section 3.6.2 set to 10, and ℓ set to $k + 2$.

All the experiments were conducted in single-thread mode, on a 2.3 GHz Intel Xeon E5 CPU, 128 GB RAM. A time limit of two hours of computation and a memory limit of 10 GB were imposed for every run.

3.6.4 Benchmark Instances

In order to test and compare the performances of the discussed methods, a set of 136 instances was arranged starting from 14 different networks collected from the literature.

For each network, several combinations of values for k and b were selected by reasoning on the core number distribution of the nodes. Table 3.1 reports, for each network, the bibliographic source from which it was collected, the number of its nodes, the number of its edges, the core number distribution of its nodes with respect to the selected values for k, as well as the associated sizes of the network after pre-processing and, finally, the selected values for the budget b.

network	#nodes	#edges	k	#nodes after pre-processing	#edges after pre-processing	budget
adjnoun [New06]	112	425	5	63	298	{3}
			4	79	359	{3, 4, 5}
			3	89	389	{3, 4, 5}
			2	102	415	{3, 4, 5}
as-22july06 [Uni04]	22963	48436	15	168	3115	{3, 4, 5}
			10	322	4845	{3, 4, 5}
			5	1087	9493	{3, 4, 5}
astro-ph [New01]	16706	121251	42	400	10552	{3, 4, 5}
			32	936	23433	{3, 4, 5}
			28	1393	32375	{3, 4, 5}
cond-mat [New01]	16726	47594	9	943	6573	{3, 4, 5}
			8	1487	9544	{3, 4, 5}
			7	2227	13280	{3, 4, 5}
			6	3442	18713	{3, 4, 5}
cond-mat-2003 [New01]	31163	120029	13	1132	12732	{3, 4, 5}
			12	1609	17327	{3, 4, 5}
			10	2901	28339	{3, 4, 5}
			9	4071	36920	{3, 4, 5}
cond-mat-2005 [New01]	40421	175692	14	1793	24595	{3, 4, 5}
			13	2151	28640	{3, 4, 5}
			12	2808	35214	{3, 4, 5}
			11	3555	42346	{3, 4, 5}
dolphins [LSB+03]	62	159	4	36	109	{3}
			3	45	135	{3, 4, 5}
			2	53	150	{3, 4, 5}
football [GN02]	115	613	8	114	606	{3, 4, 5}
			7	115	613	{3, 4, 5}
hep-th [New01]	8361	15751	7	137	885	{3, 4, 5}
			6	358	1847	{3, 4, 5}
			5	851	3775	{3, 4, 5}
			4	1735	6552	{3, 4, 5}
karate [Zac77]	34	78	2	33	77	{3, 4, 5}
lesmis [Knu93]	77	254	6	38	186	{3, 4, 5}
			4	41	197	{3, 4, 5}
			3	48	215	{3, 4, 5}
			2	59	236	{3, 4, 5}
netscience [New06]	1589	2742	5	247	976	{3, 4, 5}
			4	470	1511	{3, 4, 5}
			3	751	2045	{3, 4, 5}
			2	1141	2535	{3, 4, 5}
polbooks [Kre99]	105	441	5	65	300	{3, 4}
			4	98	422	{3, 4, 5}
			3	103	437	{3, 4, 5}
			2	105	441	{3, 4, 5}
power [WS98]	4941	6594	4	36	106	{3, 4, 5}
			3	231	479	{3, 4, 5}
			2	3353	5006	{3, 4, 5}

Table 3.1: Detailed description of the instance set.

3.6.5 Effectiveness of the Collapsed k-Core formulations

In the following, we will report some summary tables and charts which let us analyse and compare the tested methods. Because of the imposed memory limits, the Time-Dependent Model can be solved only for 87 instances while, for the remaining 48, even the relaxation at the root node is not solved to optimality. For this reason, we summarize in Table 3.2 the results obtained by testing the four methods on such subset of 87 instances, reporting for each of them: #opt, the number of optimal solutions found by the method within the limits; LB, the average lower

bound; UB, the average upper bound; gap[%], the average percentage gap, where the gap is calculated as $\frac{100*(\text{UB}-\text{LB})}{\text{UB}}$ per each instance; time[s], the average time in seconds; B&C nodes, the number of nodes of the branch-and-cut tree used to solve the models; LB_r, the average lower bound computed by solving the relaxation at the root node; gap_best[%], the average gap w.r.t. the best known solution (dimension of the k-core), calculated as $\frac{100*(\text{UB}-\text{UB}_\text{best})}{\text{UB}_\text{best}}$ per each instance; gap_r[%], the average percentage gap w.r.t. the root bound, calculated as $\frac{100*(\text{UB}-\text{LB}_\text{r})}{\text{UB}}$ per each instance.

	#opt	LB	UB	gap[%]	time[s]	B&C nodes	LB_r	gap_best[%]	gap_r[%]
Time-Dependent Model	**34**	84.6	207.0	41.2	5234	79084	68.7	4.66	71.5
Sparse Model	**42**	98.2	207.4	32.2	4081	30432	81.6	1.91	70.8
Nonlinear Model	**52**	112.3	201.7	20.4	3347	1110881	81.6	0.38	70.4
Bilevel Solver	**26**	23.1	212.1	55.8	5297	18263	2.67	4.68	96.5

Table 3.2: Summary of the computational performances of the four exact methods on the set of 87 instances solved by all the approaches.

The results show a clear superiority of the Nonlinear Model, both in terms of time and solution quality. Indeed, the Nonlinear Model provides the highest number of optimal solutions among the tested methods, yielding 52 out of 87 instances solved to optimality. Furthermore, the average computing time required by the Nonlinear Model is considerably lower than the amount of time required by the other formulations; accordingly, the provided average final lower and upper bounds values and gaps are tighter. On average, the number of nodes explored by the branch-and-cut approach solving the Nonlinear Model is greater than the number of nodes explored by the one solving the Sparse Model and the number of nodes explored by the Bilevel Solver. This reflects the fact that the problems considered at each node of the branch-and-cut tree solving the Nonlinear Model are easier to solve w.r.t. the ones of the other models, so that in the same amount of time, more nodes are explored. All the three proposed problem-specific methods exhibit better performing behaviors than the general purpose Bilevel Solver.

Since the imposed memory limits prevented the Time-Dependent Model from solving the remaining instances, from now on we restrict the comparison to the other three methods and consider the whole instance set described in Subsection 3.6.4. In particular, in Table 3.3, we report the same information as in Table 3.2, but this time for the whole set of 136 instances, with respect to the three following methods: Sparse Model, Nonlinear Model and Bilevel Solver.

	#opt	LB	UB	gap[%]	time[s]	B&C nodes	LB_r	gap_best[%]	gap_r[%]
Sparse Model	**47**	264.0	911.4	47.5	5205	21871	253.3	1.89	72.1
Nonlinear Model	**57**	274.0	892.8	39.6	4735	744905	253.3	0.04	71.6
Bilevel Solver	**26**	22.4	915.9	71.3	5984	11892	3.80	3.76	97.7

Table 3.3: Summary of the computational performances of Sparse Model, Nonlinear Model and Bilevel Solver on the whole set of 136 instances.

The results on the whole set of instances confirm the computational dominance of the Nonlinear Model, which solves 57 out of the 136 instances to optimality and almost always provides solutions values which are better or equal to the ones found by the other methods, with an average gap of 0.04%, computed with respect to the best known feasible solutions. Again, the number of branch-and-cut nodes reflects the faster resolution of the continuous relaxation of the Nonlinear Model at branching nodes.

We further provide three summary charts related to the three methods solving all the 136 instances. The first chart, shown in Figure 3.7, reports the number of instances solved to optimality within a given computational time. The second one, in Figure 3.8a, shows the optimality gap at termination, i.e., what we called gap[%]. In particular, the plot shows the number of instances (on the vertical axis) for which the gap at termination is smaller or equal than the value reported on the horizontal axis. Figure 3.8b reports the gap between the feasible solution at termination, and the best found feasible collapsed k-core among the three compared approaches, i.e., what we called gap_{best}[%]. Again, the chart shows the number of instances (on the vertical axis) for which the value of gap_{best} is smaller or equal than the value reported on the horizontal axis.

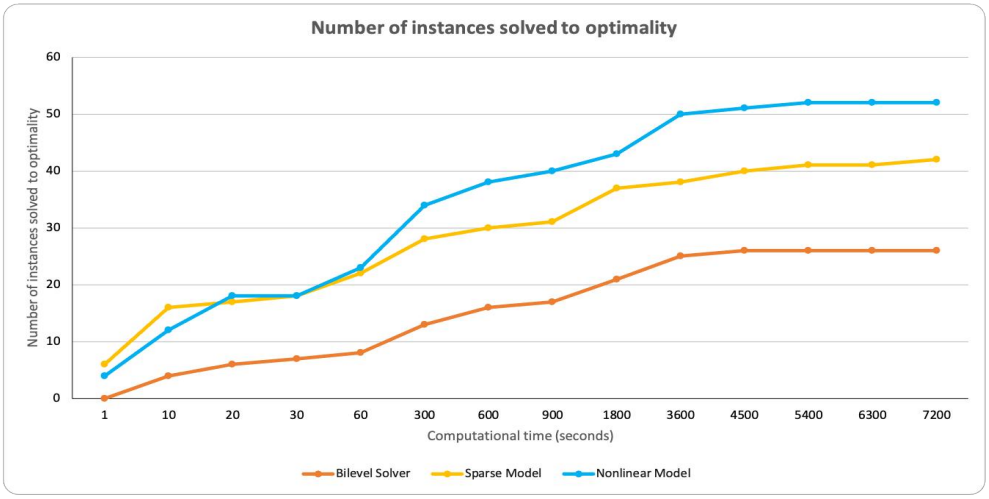
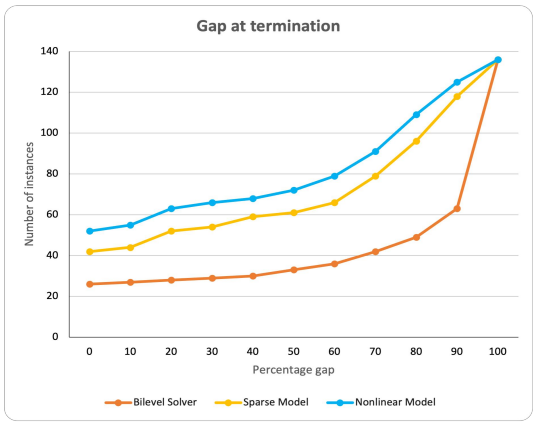
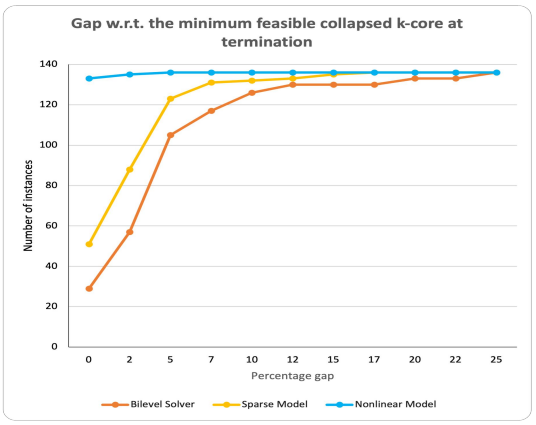


Figure 3.7: Number of instances solved to optimality within a certain computational time.



(a) Cumulative chart of percentage gap at termination.



(b) Cumulative chart of percentage gap with respect to the best feasible k-core at termination.

Figure 3.8: Cumulative charts of two different percentage gaps.

Overall, all the three charts show that the two approaches proposed in this paper are much more effective than the bilevel solver, which is largely outperformed by each of them. In the first chart (Figure 3.7), it can be observed that, only when the computational time is strictly below 20 seconds, the number of instances solved to optimality by the Sparse Model is slightly greater than the number of instances solved to optimality by the Nonlinear Model. However, when the considered time is larger than 20 seconds, the Nonlinear Model dominates the other two approaches. The chart in Figure 3.8a demonstrates that the Nonlinear Model produces a gap at termination which is always lower than the one returned by the other approaches. Finally, Figure 3.8b shows that, for more than 130 instances out of 136, the Nonlinear Model produces the best feasible solution, and, for the remaining 4 instances, the gap w.r.t. the best feasible solution found by one of the other models is below 3%. As regards Sparse Model, it returns the best feasible solution for about 50 instances, and the gap for the remaining instances is less than 17%. Finally, the bilevel solver finds the best feasible solution only for about 30 instances out of 136, and a gap w.r.t. the best one that can be as high as 25%.

Finally, we provide two sensitivity analysis tables in which we group instances into three groups: *Small* (with $n \leq 100$), *Medium* (with $100 < n \leq 1000$) and *Large* (with $n > 1000$), where n is the number of nodes of the network after pre-processing procedures. For each group, we consider three values of the budget $b \in \{3, 4, 5\}$, thus obtaining nine classes of instances. For each class, we report in both Tables 3.4 and 3.5 the following values:#total, the total number of instances in the class; $\mathrm{LB_c}$, the average of the combinatorial lower bounds m (see Section 3.5.3) that we precalculate. Moreover, in Table 3.4, for each method we report some common exploration indicators, such as the number of instances solved to optimality (#opt), the average computing time in seconds (time[s]) and the average number of nodes of the branch-and-cut tree (B&C nodes); additionally, for the Sparse Model we also report the average number of added cuts. In Table 3.5, instead, we report for each class gap[%], $\mathrm{gap_r}$[%], and $\mathrm{gap_{best}}$[%]. The re-

ported values show how increasing the budget affects the different methods. As expected, when the value of b increases, the number of instances solved to optimality decreases for each method, while the average resolution time and the average gap value increase. Specifically, all the instances of the first class, i.e., small instances with $b = 3$, are solved to optimality by both the Sparse and Nonlinear models, while the Bilevel Solver provides the optimal certified solution for all except two of them. The Nonlinear Model solves to optimality also all the instances of the second class, i.e., small instances with $b = 4$. For the remaining classes, instead, no method is able to certify the optimality of all the instances from a given subclass. In particular, no optimal solution is found for any of the instances from the group "large", within the imposed time limit. Overall, this analysis shows that the difficulty of an instance is highly affected by the budget value, in addition to the network size, but, for non-large instances, the Nonlinear Model confirms its superiority w.r.t. the remaining approaches.

Class details				Sparse Model				Nonlinear Model			Bilevel Solver		
size	**b**	**#total**	**LB_c**	**#opt**	**time[s]**	**B&C nodes**	**cuts**	**#opt**	**time[s]**	**B&C nodes**	**#opt**	**time[s]**	**B&C nodes**
Small	3	14	11	14	102	5481	1719	14	27.6	26633	12	1623	10455
	4	12	6	11	1296	31205	5409	12	242	332382	7	3574	32115
	5	11	4	6	3820	53377	9226	10	1413	1960571	4	4805	45443
Medium	3	17	139	7	4633	23344	16707	9	3961	353716	1	6884	11410
	4	17	117	3	6431	34573	19101	6	5544	1381271	1	6863	10923
	5	17	97	1	6777	39951	16420	1	6938	2460756	1	6817	10669
Large	3	16	776	0	7200	5750	11795	0	7200	90644	0	7200	573
	4	16	566	0	7200	4393	9417	0	7200	80239	0	7200	492
	5	16	419	0	7200	6881	6206	0	7200	82350	0	7200	474

Table 3.4: Sensitivity analysis showing the effect of different budget values on the number of instances solved to optimality, the resolution time and other exploration indicators.

Class details				Sparse Model			Nonlinear Model			Bilevel Solver		
size	**b**	**#Total**	**LB_c**	**gap[%]**	**gap_r[%]**	**gap_{best}[%]**	**gap[%]**	**gap_r[%]**	**gap_{best}[%]**	**gap[%]**	**gap_r[%]**	**gap_{best}[%]**
Small	3	14	11	0.00	75.1	0.00	0.00	75.1	0.00	7.40	92.5	0.28
	4	12	6	6.40	82.4	0.11	0.00	82.4	0.00	20.2	95.6	1.28
	5	11	4	34.5	87.7	1.40	3.10	87.7	0.00	34.4	97.2	2.19
Medium	3	17	139	31.2	57.3	0.46	22.6	57.1	0.17	78.0	97.0	5.35
	4	17	117	50.5	65.5	2.52	34.2	64.9	0.12	84.2	97.9	6.89
	5	17	97	66.5	73.3	3.43	56.5	72.5	0.00	85.8	99.0	7.87
Large	3	16	776	64.0	64.0	2.14	63.0	63.1	0.00	98.8	99.7	2.06
	4	16	566	73.4	73.4	2.69	72.3	72.4	0.00	99.1	99.8	2.62
	5	16	419	79.8	79.8	3.38	78.9	78.9	0.00	99.5	99.9	3.22

Table 3.5: Sensitivity analysis showing the effect of different budget values on the gaps.

Chapter 4

Cluster Deletion Problem

Clustering consists in partitioning a given set into disjoint subsets of items, by considering the similarities between them and producing homogeneous and well-separated clusters. The graph theoretic approach to clustering involves creating a similarity graph, whose nodes are associated to the items to be clustered and such that two nodes of the graph are linked by an edge if and only if the similarity between the related items is higher than a predefined threshold [HJ97]; clusters of nodes are then identified with respect to such graph. The known class of *edge modification problems* [NSS01] studies how to perform as few modifications as possible on the edge set of a graph in order to satisfy a given property Π. Such modifications typically include adding, deleting or replacing edges. When the constraint Π is to obtain a *cluster graph*, namely a disjoint union of cliques, we talk about *cluster graph modification problems* [RRD04].

The Cluster Deletion (CD) problem, which consists in determining the minimum number of edges whose removal from a graph produces a cluster graph, is addressed. Since each node is connected to all the nodes belonging to its same cluster in any feasible solution to the problem, this setting privileges homogeneity over separation of clusters. Beyond typical clustering scenarios, the problem finds application in several areas, including wireless sensor networks [MGKP09], where the nodes hosting sensor devices are often grouped into clusters to match scalability and efficiency requirements, and bioinformatics, in the context of the detection of remote homologues of protein sequences [PSS+02]. In the last scenario, proteins are generally compared by aligning their amino acid sequences with techniques such as BLAST and FASTA [KV98], which rely on the computation of similarity matrices. Such matrices are exploited to identify groups of homologous proteins, namely proteins descending from the same ancestor in the evolution hierarchy. Indeed, high similarity values are associated with proteins sharing the same ancestor and cluster analysis is crucial to classify *cross-domain* proteins, which simultaneously belong to more functional groups. Interesting applications also arise in DNA clone classification scenarios, where the cluster analysis of fingerprint data is a key step in the application of DNA array-based fingerprinting methods, used to characterize cDNA and rDNA [FBJ04]. More in detail, using control DNA clones, fingerprint data are first normalized and binarized. Then, the clustering of binary oligonucleotide fingerprints is modeled as a CD problem, whose resolution allows to

resolve the missing values resulting from the binarization process.

The complexity of the CD problem has been extensively studied. On generic graphs, the problem has been proved to be NP-complete and NP-hard to approximate to within some constant factor [RRD04]. When the number of clusters to be contained in a solution is fixed a priori, the problem becomes polynomially solvable with two clusters, while it is still NP-complete with three or more clusters. Furthermore, a dichotomy based on the maximum degree of the graph has been identified in [KU12]: solving the problem on graphs with maximum degree three requires polynomial time, but the problem is NP-hard even on graphs with maximum degree four.

Iteratively finding maximum cliques in the graph yields a 2-approximated solution to the optimal cluster deletion on generic graphs [DAE+07]. Although, in general, the maximum clique problem is NP-hard, it is polynomially solvable on some specific graph classes. By exploiting this observation, it has been shown that the optimal solution of the CD problem can be found in polynomial time on cographs [GHN13]. Additional polynomial-time solvable graph classes include split graphs, proper interval graphs and block graphs [BDVP15], as well as graphs that do not contain butterfly and diamond subgraphs [MN19].

The CD problem has been also examined in the context of parameterized complexity. In particular, it is classified as fixed-parameter tractable with respect to the number of edge deletions [BBBT09, Bö12, BBK11], as well as to the cluster vertex deletion number of the input graph, which is the number of nodes to be removed from the graph to produce a cluster graph [KU11, Uhl11]. Moreover, the problem is fixed-parameter tractable also with respect to the combination of the maximum number of clusters allowed in the cluster graph and the local modification bound, namely the maximum number of edge deletions affecting a single node of the graph [KU12].

It is known that minimizing the total number of edges between the clusters is equivalent to maximizing the total number of edges within the clusters [DAE+07]. In this work, we provide two mathematical formulations for the CD problem, suitable for finding in reasonable time the optimal solution when dealing with small-size networks. Furthermore, we propose a heuristic algorithm to efficiently solve larger instances of the problem. The proposed approaches are tested and compared on both real and artificially generated networks.

4.1 Notation and problem statement

Given an undirected graph $G = (V, E)$, being V and E the set of nodes and the set of edges of G, respectively, we denote by $N(i)$ the set of neighbours of node i, that is the set of nodes $j \in V$ such that $(i, j) \in E$, and by $\delta(i)$ the set of edges incident to node i.

A *clique* is a set of pairwise adjacent nodes, therefore a subset of nodes $S \subseteq V$ is a clique in G if and only if any couple of nodes $i, j \in S$, s.t. $i \neq j$, is linked by and edge of G, i.e. $(i, j) \in E$. We shortly refer to a clique of size k as a k-clique. A *connected component* of G is any connected subgraph $\bar{G} = (\bar{V}, \bar{E})$ of G such that the original set of edges E does not contain any edge between a node in $\bar{V}$ and a node in $V \setminus \bar{V}$. G is a *cluster graph* if any connected component of G is a clique.

The Cluster Deletion (CD) problem, asks for the smallest subset of edges $S \subseteq E$ such that $G \setminus S$ is a disjoint union of cliques, namely a cluster graph.

Furthermore, a *partitioning* $\mathcal{Q}$ of the set of nodes V is a collection of disjoint subsets of V such that the union of such subsets is V, i.e. *(i)* $Q_1 \cap Q_2 = \forall Q_1, Q_2 \in \mathcal{Q}$; *(ii)* $\bigcup_{Q_i \in \mathcal{Q}} Q_i = V$. When discussing the heuristic algorithm in Section 4.3, we will refer to $G^{(i)}$ as the graph obtained after performing the i-th iteration of the algorithm, and to $\mathcal{Q}_{G^{(i)}}$ as the partitioning of G built from $G^{(i)}$. Finally, we will define and denote by $G[\mathcal{Q}_{G^{(i)}}]$ the cluster graph *generated* by such partition.

4.2 Mathematical formulations

Let us denote by x the vector of the decision variables of the problem, associated to the edges of G. For any arbitrary edge $(i, j) \in E$, the related x_{ij} binary variable is defined as follows:

$$x_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ belong to the same clique} \\ 0 & \text{otherwise.} \end{cases}$$

Note that, by definition, when $x_{ij} = 1$, both the endpoints of the edge are in the same clique; this directly implies the presence of the edge (i, j) in the resulting cluster graph. In the following, we introduce an Integer Linear Programming (ILP) formulation for the problem:

Formulation 1.

$$\max \sum_{(i,j) \in E} x_{ij} \tag{4.1a}$$

$$s.t.\ x_{ij} + x_{ik} \leq x_{jk} + 1 \qquad \forall\, i \in V,\ \forall j, k \in N(i) : j < k, (j, k) \in E \tag{4.1b}$$

$$x_{ij} + x_{ik} \leq 1 \qquad \forall\, i \in V,\ \forall j, k \in N(i) : j < k, (j, k) \notin E \tag{4.1c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in E. \tag{4.1d}$$

The objective function (4.1a) maximizes the number of edges belonging to the cluster graph, which is equivalent to minimizing the number of the removed edges.

Both families of constraints (4.1b) and (4.1c) work on the neighborhood $N(i)$ of each node $i \in V$. For any couple of adjacent nodes $j, k \in N(i)$, we must distinguish two cases:

- The edge (j, k) exists in the graph (i.e. $(i, k) \in E$). If it belongs to some clique (i.e. $x_{jk} = 1$), no restriction on the value of x_{ij} and x_{ik} is required. Otherwise, if (j, k) is not selected in the solution (i.e. $x_{jk} = 0$), nodes i, j and k cannot constitute a clique, and at most one between the edges (i, j) and (i, k) can be selected. Constraints (4.1b) force this requirement to be satisfied, when needed.

- The edge (j,k) does not exists in the original graph (i.e. $(i,k) \notin E$). If this is the case, the edges (i,j) and (i,k) cannot be simultaneously selected. Constraints (4.1c) require that at most one between x_{ij} and x_{ik} is equal to 1.

In order to provide a proof of correctness for the introduced ILP formulation, let us state and prove the following property.

Lemma 5. *Given three arbitrary nodes of G, let say $i,j,k \in V$, any cluster graph C of G such that $(j,k) \notin C$ contains at most only one edge between (i,j) and (i,k).*

Proof. The statement can be easily proven by contradiction. Let us suppose that there exists a cluster graph $\bar{C}$ of G such that (j,k) does not belong to $\bar{C}$ while both (i,j) and (i,k) do. Since $(i,j) \in \bar{C}$, then nodes i and j belong to the same clique. The same holds for nodes i and k, since also $(i,k) \in \bar{C}$, by hypothesis. Consequently, nodes i,j and k all belong to the same clique. However, there is no edge between nodes j and k in $\bar{C}$, thus $\{i,j,k\}$ is not a clique in $\bar{C}$ and $\bar{C}$ is not a cluster graph. □

An alternative formulation, based on the enumeration of all possible cliques of size equal to three, i.e. triangles of the graph, can be designed by exploiting the following property.

Lemma 6. *Given a 3-clique $C \subset V, |C| = 3$, the number of edges between any two nodes of C belonging to any possible cluster graph $\bar{C}$ of G is zero, one or three.*

Proof. Let $C = \{i,j,k\}$ and $\bar{C}$ be any 3-clique and cluster graph of G, respectively. Depending on the number η of edges between any two nodes of C belonging to $\bar{C}$, four cases can occur: *(i)* $\eta = 0$, if all the edges have been removed from G in order to obtain $\bar{C}$ and the nodes $i,j,k \in C$ do not belong to the same clique in $\bar{C}$; *(ii)* $\eta = 1$, when a single edge is left, and its endpoints belong to the same clique in $\bar{C}$, while the remaining node does not; *(iii)* $\eta = 2$, i.e. exactly two edges are left, which leads to a contradiction because i,j and k cannot constitute a clique in $\bar{C}$; *(iv)* meaning that no edge is removed and the original 3-clique is preserved in $\bar{C}$. □

Let us denote by C_3 the set of triangles (3-cliques) in G:

$$C_3 = \{S \subset V : |S| = 3, (i,j) \in E, \ \forall\, i,j \in S, i \neq j\}$$

In addition to the previously defined x_{ij} decision variables, associated to the edges of G, we introduce the y vector of binary auxiliary variables, defined for each triangle $\{i,j,k\} \in C_3$ of G, as follows:

$$y_{ijk} = \begin{cases} 1 & \text{if } x_{ij} + x_{ik} + x_{jk} \geq 3 \\ 0 & \text{if } x_{ij} + x_{ik} + x_{jk} \leq 1. \end{cases}$$

According to Lemma 6, $x_{ij}+x_{ik}+x_{jk} \in \{0,1,3\}$ must be required for every triangle $\{i,j,k\} \in C_3$. To this aim, $x_{ij} + x_{ik} + x_{jk}$ is constrained to be either less or equal to 1 or greater or equal to 3, when the y_{ijk} auxiliary variable related to $\{i,j,k\}$ assumes value 0 or 1, respectively. The resulting ILP formulation follows.

Formulation 2.

$$\max \sum_{(i,j)\in E} x_{ij} \tag{4.2a}$$

$$s.t.\ x_{ij} + x_{ik} + x_{jk} \leq 1 + 2y_{ijk} \qquad \forall\, \{i,j,k\} \in C_3, i < j < k \tag{4.2b}$$

$$x_{ij} + x_{ik} + x_{jk} \geq 3y_{ijk} \qquad \forall\, \{i,j,k\} \in C_3, i < j < k \tag{4.2c}$$

$$x_{ij} + x_{ik} \leq 1 \qquad \forall\, i \in V,\ \forall j,k \in N(i) : j < k, (j,k) \notin E \tag{4.2d}$$

$$y_{ijk} \in \{0,1\} \qquad \forall\, \{i,j,k\} \in C_3, i < j < k \tag{4.2e}$$

$$x_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in E. \tag{4.2f}$$

The objective function (4.2a) and the family of constraints (4.2d) specifying the edges that cannot be simultaneously selected have the same effect of (4.1a) and (4.1c) in Formulation 1, respectively. Furthermore, constraints (4.2b) and (4.2c) force the number of unremoved edges between the nodes of each triangle to be different from two.

4.3 Heuristic approach

In this section, we present a heuristic algorithm for the CD problem and formally study its complexity. The algorithm repeatedly removes an edge from the graph and collapses its endpoints into a single new node, ensuring that the resulting graph is a cluster graph.

The edge contraction operation is formally defined in Subsection 4.3.1, while the algorithm is presented, along with the related pseudo-code, in Subsection 4.3.2.

4.3.1 Edge contraction operations

The edge contraction operation consists in removing an edge from the graph and then merging the two nodes previously linked by it. Such operation is fundamental in the theory of graph minors, i.e., obtained by removing nodes and edges and by contracting edges from the original graph.

Given a simple undirected graph $G = (V, E)$ and an edge $(u, v) \in E$, contracting (u, v) on G produces a new graph $G' = (V', E')$, where:

- $V' = V \setminus \{u, v\} \cup \{\mathcal{W}\}$;
- $E' = E \setminus \{(u, z) : z \in N(u)\} \setminus \{(v, z) : z \in N(v)\} \cup \{(\mathcal{W}, z) : z \in N_G(u) \vee z \in N_G(v)\}$.

The new node $\mathcal{W}$ replaces u and v in G' and new edges incident on $\mathcal{W}$ are added to G' in place of the edges incident on u and v in G. We call $\mathcal{W}$ a *super-node*, since it represents more than a single node of the original graph. More in general, we define a super-node as a subset of nodes

of the original graph, i.e., $\mathcal{W} \in \mathcal{P}(V)$, and we use calligraphic letters to denote super-nodes. According to this definition, every singleton of a node in V is a super-node.

As a result of the edge contraction operation, the creation of multi-graphs can be allowed or not. In our case, there not exist multiple edges between any two nodes. This means that, when contracting (u, v) in $\mathcal{W}$, if there is a node $z \in V$ such that both $(u, z) \in E$ and $(v, z) \in E$, only a single edge replaces them in the resulting graph, i.e. $(\mathcal{W}, z) \in E'$. However, for each edge $(\mathcal{U}, \mathcal{V}) \in E'$, with $\mathcal{U}$ and $\mathcal{V}$ super-nodes, we store the number of edges of the original graph which connect a node in $\mathcal{U}$ with a node in $\mathcal{V}$.

4.3.2 The algorithm

In order to produce a cluster graph, the proposed algorithm iteratively performs edge contraction operations on *contractible* edges. By doing so, at each iteration of the algorithm, the super-nodes represent a partitioning of V into disjoint cliques.

Definition 1. *An edge $(\mathcal{U}, \mathcal{V})$, with $\mathcal{U}, \mathcal{V} \in \mathcal{P}(V)$ super-nodes, is said to be contractible if and only if $\mathcal{U} \cup \mathcal{V}$ is a clique in G.*

Algorithm 4 shows the simple high-level pseudo-code of the proposed approach. The procedure takes as input the original graph G. At the beginning of the computation (line 1), the set E_t of contractible edges is set equal to the whole set of edges of the original graph. In the main cycle (lines 2-5), the algorithm iteratively selects a contractible edge $(\mathcal{U}, \mathcal{V})$ from E_t and performs the contraction operation, updating the current graph G and the set of contractible edges. Finally, the set V of super-nodes of the last current graph is returned.

Algorithm 4: ECHeuristic

Input: The original graph $G = (V, E)$

1 $E_t \leftarrow E$
2 **while** $|E_t| > 0$ **do**
3 $\quad (\mathcal{U}, \mathcal{V}) \leftarrow$ a contractible edge from E_t
4 $\quad G \leftarrow G/_{(\mathcal{U},\mathcal{V})}$
5 $\quad E_t \leftarrow$ contractible edges in G
6 **return** V

The execution of `ECHeuristic` on a graph G identifies a sequence of graphs $G^{(0)}, G^{(1)}, \ldots, G^{(p)}$, where $G^{(0)} = G$ and $G^{(i)} = (V^{(i)}, E^{(i)})$ is the current graph at the end of the i-th iteration.

Observation 1. *The number p of iterations performed by `ECHeuristic` on a graph G is upper bounded by the number of nodes in G minus one, i.e. $1 \leq p \leq |V| - 1$.*

Starting from any of the graphs $G^{(i)}, \forall\ i \in \{0, \ldots, p\}$, it is possible to build a partitioning $\mathcal{Q}_{G^{(i)}}$ of the set V of nodes of the original graph G, in which the nodes in the same super-node

are assigned to the same partition. Formally, $\mathcal{Q}_{G^{(i)}} = \{\mathcal{V} : \mathcal{V} \in V^{(i)}\}$. Given a node $v \in V$, let us denote by $\mathcal{Q}_{G^{(i)}}(v)$ the partition of $\mathcal{Q}_{G^{(i)}}$ containing v. By removing from G all the edges $(u, v) \in E$ such that $\mathcal{Q}_{G^{(i)}}(u) \neq \mathcal{Q}_{G^{(i)}}(v)$, that is the edges whose endpoints are assigned to different partitions of $\mathcal{Q}_{G^{(i)}}$, a spanning subgraph of G is obtained. We denote such subgraph by $G[\mathcal{Q}_{G^{(i)}}]$ and say that $\mathcal{Q}_{G^{(i)}}$ *generates* $G[\mathcal{Q}_{G^{(i)}}]$.

Lemma 7. *All the partitionings* $\{\mathcal{Q}_{G^{(i)}}\}_{i\in\{0,\dots,p\}}$ *of the set of nodes produced by executing* `ECHeuristic` *on a graph* G *generate cluster graphs.*

Proof. Let $G[\mathcal{Q}_{G^{(i)}}]$ be the subgraph generated by the partitioning $\mathcal{Q}_{G^{(i)}}$ associated to the i-th iteration of the algorithm. We show, by induction, that $G[\mathcal{Q}_{G^{(i)}}]$ is a cluster graph, for all $i \in \{0, 1, \dots, p\}$. Clearly, since $|\mathcal{Q}_{G^{(0)}}| = |V|$, each partition is a singleton and all the edges are removed, then $G[\mathcal{Q}_{G^{(0)}}]$ is a trivial cluster graph. On the other hand, let us assume that $G[\mathcal{Q}_{G^{(i-1)}}]$ is a cluster graph, meaning that every partition in $\mathcal{Q}_{G^{(i-1)}}$ is a clique in G, with $i \in \{1, \dots, p\}$. The partitioning produced at iteration i can be expressed in terms of the one produced at iteration $i-1$, like $\mathcal{Q}_{G^{(i)}} = \mathcal{Q}_{G^{(i-1)}} \setminus \{\mathcal{U}, \mathcal{V}\} \cup \mathcal{W}$, where $(\mathcal{U}, \mathcal{V})$ is the edge contracted at iteration i and $\mathcal{W} = \mathcal{U} \cap \mathcal{V}$ is the new partition. From Definition 1, every node in $\mathcal{U}$ is linked by an edge in E with every node in $\mathcal{V}$, otherwise $(\mathcal{U}, \mathcal{V})$ would not be contractible. As a consequence, $\mathcal{W}$ is a clique in G, thus every partition in $\mathcal{Q}_{G^{(i)}}$ is a clique in G. It follows that $G[\mathcal{Q}_{G^{(i)}}]$ is a cluster graph. □

4.3.3 Determining the set of contractible edges

Let us observe that the edge contraction operation modifies the graph locally: when contracting an edge, only the part of the graph related to its endpoints and their neighbors is affected. Starting from this intuition, we deduce an efficient way to keep track of the contractible edges throughout the iterations of the algorithm.

When an edge $(\mathcal{U}, \mathcal{V})$ is contracted on a graph $G = (V, E)$, with respect to a third super-node $\mathcal{Z} \in N(\mathcal{U}) \cup N(\mathcal{V})$, three cases can occur:

1. both $\mathcal{U}$ and $\mathcal{V}$ are connected to $\mathcal{Z}$ in G, i.e. $(\mathcal{U}, \mathcal{Z}), (\mathcal{V}, \mathcal{Z}) \in E$;
2. $\mathcal{U}$ is connected to $\mathcal{Z}$ in G, while $\mathcal{V}$ is not, i.e. $(\mathcal{U}, \mathcal{Z}) \in E$, $(\mathcal{V}, \mathcal{Z}) \notin E$;
3. $\mathcal{V}$ is connected to $\mathcal{Z}$ in G, while $\mathcal{U}$ is not, i.e. $(\mathcal{U}, \mathcal{Z}) \notin E$, $(\mathcal{V}, \mathcal{Z}) \in E$;

In case (1), $(\mathcal{U}, \mathcal{Z})$ and $(\mathcal{V}, \mathcal{Z})$ are replaced by $(\mathcal{W}, \mathcal{Z})$, where $\mathcal{W} = \mathcal{U} \cup \mathcal{V}$ is the newly introduced super-node; the weight of such edge, indicating the number of edges of the original graph with an endpoint in $\mathcal{W}$ and the other in $\mathcal{Z}$, is the sum of the weights of the original edges, i.e. $w(\mathcal{W}, \mathcal{Z}) = w(\mathcal{U}, \mathcal{Z}) + w(\mathcal{V}, \mathcal{Z})$. In case (2), $w(\mathcal{W}, \mathcal{Z}) = w(\mathcal{U}, \mathcal{Z})$, since no node in $\mathcal{V}$ is connected to any node in $\mathcal{Z}$. Similarly, in case (3), $w(\mathcal{W}, \mathcal{Z}) = w(\mathcal{V}, \mathcal{Z})$.

In the resulting graph $G/_{(\mathcal{U},\mathcal{V})}$, the contractibility of an edge can be determined by checking its weight. Indeed, by Definition 1, an edge is contractible iff its endpoints form a clique in G. An equivalent characterization, in terms of the weights of the edge, is given in Definition 2.

Definition 2. *An edge $(\mathcal{U},\mathcal{V})$, with $\mathcal{U},\mathcal{V} \in \mathcal{P}(V)$ super-nodes, is said to be contractible if and only if $w(\mathcal{U},\mathcal{V}) = |\mathcal{U}||\mathcal{V}|$.*

Let us denote by $E_t^{(i)}$ and $E_f^{(i)}$ the sets of contractible and non-contractible edges at iteration i, respectively. Note that $E_f^{(i)} = E^{(i)} \setminus E_t^{(i)}$.

Observation 2. *For $i = 1,\dots,p$, $|E_t^{(i)}| < |E_t^{(i-1)}|$, while $|E_f^{(i)}| \geq |E_f^{(i-1)}|$.*

Observation 3. *The cluster graph identified by the algorithm is obtained by removing the edges in $E_f^{(p)}$ from the original graph G. As a result, the number of edges that need to be removed from G in order to obtain such cluster graph is given by $\sum_{(\mathcal{U},\mathcal{V})\in E_f^{(p)}} w(\mathcal{U},\mathcal{V})$.*

Lemma 8. *After the contraction of an edge $(\mathcal{U},\mathcal{V})$ in $\mathcal{W} = \mathcal{U}\cup\mathcal{V}$, an edge $(\mathcal{W},\mathcal{Z}), \mathcal{Z} \in N^{(i-1)}(\mathcal{U}) \cap N^{(i-1)}(\mathcal{V})$ is contractible at iteration i if and only if the edges $(\mathcal{U},\mathcal{Z})$ and $(\mathcal{V},\mathcal{Z})$ were contractible at iteration $i-1$, i.e. $(\mathcal{W},\mathcal{Z}) \in E_t^{(i)} \iff (\mathcal{U},\mathcal{Z}),(\mathcal{V},\mathcal{Z}) \in E_t^{(i-1)}$.*

Proof. Let us assume that $(\mathcal{W},\mathcal{Z})$ is contractible at iteration i, i.e. $(\mathcal{W},\mathcal{Z}) \in E_t^{(i)}$. Definition 1 implies that $\mathcal{W}\cup\mathcal{Z}$ is a clique in G. This means that every node in $\mathcal{W}$ is linked with every node in $\mathcal{Z}$, i.e. $(w,z) \in E, \forall\, w \in \mathcal{W}, \forall\, z \in \mathcal{Z}$. Since $\mathcal{W} = \mathcal{U}\cup\mathcal{V}$, the same holds for any couple of nodes in $\mathcal{U}\times\mathcal{Z}$ and $\mathcal{V}\times\mathcal{Z}$, respectively. As a consequence, $(\mathcal{U},\mathcal{Z})$ and $(\mathcal{V},\mathcal{Z})$ are both contractible at iteration $i-1$, i.e. $(\mathcal{U},\mathcal{Z}),(\mathcal{V},\mathcal{Z}) \in E_t^{(i-1)}$.

On the other hand, if $(\mathcal{U},\mathcal{Z}),(\mathcal{V},\mathcal{Z}) \in E_t^{(i-1)}$, from Definition 2 follows that $w(\mathcal{U},\mathcal{Z}) = |\mathcal{U}||\mathcal{Z}|$ and $w(\mathcal{V},\mathcal{Z}) = |\mathcal{V}||\mathcal{Z}|$. When contracting $(\mathcal{U},\mathcal{V})$ at iteration i, since $\mathcal{Z} \in N(\mathcal{U}) \cap N(\mathcal{V})$, $w(\mathcal{W},\mathcal{Z}) = w(\mathcal{V},\mathcal{Z}) + w(\mathcal{V},\mathcal{Z}) = |\mathcal{U}||\mathcal{Z}| + |\mathcal{V}||\mathcal{Z}| = (|\mathcal{U}| + |\mathcal{V}|)|\mathcal{Z}| = |\mathcal{W}||\mathcal{Z}|$. By Definition 2, $w(\mathcal{W},\mathcal{Z}) = |\mathcal{W}||\mathcal{Z}|$ means that $(\mathcal{W},\mathcal{Z})$ is contractible at iteration i, i.e. $(\mathcal{W},\mathcal{Z}) \in E_t^{(i)}$. □

By Lemma 8, the set of contractible edges at iteration i can be efficiently determined starting from the set of contractible edges at iteration $i-1$, as follows:

$$E_t^{(i)} = E_t^{(i-1)} \setminus E(\mathcal{U}) \setminus E(\mathcal{V}) \cup \{(\mathcal{W},\mathcal{Z}) : \mathcal{Z} \in N^{(i-1)}(\mathcal{U}) \cap N^{(i-1)}(\mathcal{V})\}$$

Indeed, the edges with no endpoint in $\{\mathcal{U},\mathcal{V}\}$, which were contractible at iteration $i-1$, are not affected by the contraction of the edge $(\mathcal{U},\mathcal{V})$. After removing from $E_t^{(i-1)}$ the edges removed by the contraction operation, in order to obtain the new set of contractible edges $E_t^{(i)}$, it is sufficient to add the newly added edges incident on $\mathcal{W}$ whose other endpoint was incident on both $\mathcal{U}$ and $\mathcal{V}$.

4.3.4 Choosing the next contractible edge

By Observation 3, the value of a solution can be expressed as $\sum_{(\mathcal{U},\mathcal{V})\in E_f^{(p)}} w(\mathcal{U},\mathcal{V})$, where p is the last iteration of the algorithm. Based on such observation, we define a greedy criterion for the choice of the contractible edge at each iteration. Given the sets of contractible and non-contractible edges $E_t^{(i)}$ and $E_f^{(i)}$ at iteration i, we select among the contractible edges $E_t^{(i)}$ the one whose contraction minimizes the number of non-contractible edges at iteration $i+1$. More in detail, the edge to contract at iteration i is identified by:

$$\arg\min_{(\mathcal{U},\mathcal{V})\in E_t^{(i)}} |E_f^{(i+1)} / (\mathcal{U},\mathcal{V})|.$$

4.4 Computational experiments

In order to extensively test and evaluate the performance of the proposed approaches, we generated a set of 120 instances according to the Barabási–Albert model [BA99, Bar09], which relies on preferential attachment and growth mechanisms to obtain random scale-free networks characterized by relatively few nodes with unusually high degree with respect to the remaining nodes. More in detail, the procedure iteratively adds new nodes and connections to the network: the more the connections of a node, the higher the probability of establishing new connections in which such node is involved as the network enlarges. Furthermore, a smaller set of real benchmark networks [RA15] was also considered in the test phase, in order to analyze the behavior of the proposed approach on biological networks, where the problem finds relevant application.

All the experiments have been conducted on the same machine, equipped with a 2.3 GHz Intel Xeon E5 CPU and 128 GB of RAM. The three proposed solution approaches have been implemented in Python and, in particular, the two mathematical models have been solved by using the IBM ILOG CPLEX 12.10.0 solver and imposing a computation time limit of one hour.

4.4.1 Barabási–Albert networks

By adopting the Barabási–Albert scheme, we generated instances of several sizes and densities. In particular, $n \in \{100, 200, 400, 600, 800, 1\,000\}$ and $d \in \{2\%, 4\%, 6\%, 8\%\}$, where d is the ratio between the actual number m of edges in the network and the total number of potential edges, i.e. $d = 2m/(n(n-1))$. For each combination of n and d values, we generated 5 instances, by obtaining 24 different scenarios.

Tables 4.1 and 4.2 report detailed information about the performances of the proposed approaches on each of the instances with a number of nodes up to 400 and starting from 600, respectively. More in detail, the number of nodes n, the number of edges m and the density d of a given network are reported in the associated row of the first three columns, while the fourth column reports random seeds which uniquely identify the instances, since they are used in

					Formulation 1				Formulation 2				EC Heuristic		
Nodes	Edges	Density	Seed	Best	Time	Status	Value	Gap	Time	Status	Value	Gap	Time	Value	Gap
100	99	2%	1531	**66**	0,00172	101	66	**0,00%**	0,00168	101	66	**0,00%**	0,00685	66	**0,00%**
100	99	2%	1561	**63**	0,00236	101	63	**0,00%**	0,00162	101	63	**0,00%**	0,00455	63	**0,00%**
100	99	2%	4146	**64**	0,00187	101	64	**0,00%**	0,00194	101	64	**0,00%**	0,00932	64	**0,00%**
100	99	2%	8899	**65**	0,00184	101	65	**0,00%**	0,00190	101	65	**0,00%**	0,00577	65	**0,00%**
100	99	2%	9323	**66**	0,00184	101	66	**0,00%**	0,00182	101	66	**0,00%**	0,00679	66	**0,00%**
100	196	4%	1018	**154**	0,08356	101	154	**0,00%**	0,01296	101	154	**0,00%**	0,01260	154	**0,00%**
100	196	4%	2785	**150**	0,00898	101	150	**0,00%**	0,01049	101	150	**0,00%**	0,01200	152	1,33%
100	196	4%	3894	**143**	0,00898	101	143	**0,00%**	0,01153	101	143	**0,00%**	0,01213	143	**0,00%**
100	196	4%	6356	**147**	0,00898	101	147	**0,00%**	0,01010	101	147	**0,00%**	0,01132	148	0,68%
100	196	4%	9890	**152**	0,00972	101	152	**0,00%**	0,01078	101	152	**0,00%**	0,02095	152	**0,00%**
100	291	6%	1192	**227**	0,01754	101	227	**0,00%**	0,02055	101	227	**0,00%**	0,03116	231	1,76%
100	291	6%	3706	**227**	0,03633	101	227	**0,00%**	0,02995	101	227	**0,00%**	0,02616	229	0,88%
100	291	6%	8400	**235**	0,02043	101	235	**0,00%**	0,03233	101	235	**0,00%**	0,03563	236	0,43%
100	291	6%	9236	**228**	0,01533	101	228	**0,00%**	0,01796	101	228	**0,00%**	0,02008	232	1,75%
100	291	6%	9578	**233**	0,03536	101	233	**0,00%**	0,03447	101	233	**0,00%**	0,02084	237	1,72%
100	384	8%	1364	**307**	0,08794	101	307	**0,00%**	0,13286	101	307	**0,00%**	0,04241	311	1,30%
100	384	8%	3566	**316**	0,07926	101	316	**0,00%**	0,11339	101	316	**0,00%**	0,02927	322	1,90%
100	384	8%	4040	**311**	0,03348	101	311	**0,00%**	0,03291	101	311	**0,00%**	0,02940	315	1,29%
100	384	8%	6016	**315**	0,03363	101	315	**0,00%**	0,09355	101	315	**0,00%**	0,03094	318	0,95%
100	384	8%	7282	**307**	0,04967	101	307	**0,00%**	0,08649	101	307	**0,00%**	0,03094	314	2,28%
200	396	2%	1999	**306**	0,03438	101	306	**0,00%**	0,02633	101	306	**0,00%**	0,02822	307	0,33%
200	396	2%	2797	**304**	0,02130	101	304	**0,00%**	0,02012	101	304	**0,00%**	0,04274	304	**0,00%**
200	396	2%	3174	**310**	0,03492	101	310	**0,00%**	0,02378	101	310	**0,00%**	0,03185	312	0,65%
200	396	2%	3761	**305**	0,01447	101	305	**0,00%**	0,01630	101	305	**0,00%**	0,02761	306	0,33%
200	396	2%	6802	**301**	0,01323	101	301	**0,00%**	0,01751	101	301	**0,00%**	0,02467	303	0,66%
200	784	4%	2309	**650**	0,22450	101	650	**0,00%**	0,23380	101	650	**0,00%**	0,07858	658	1,23%
200	784	4%	4114	**649**	0,16566	101	649	**0,00%**	0,08795	101	649	**0,00%**	0,07636	661	1,85%
200	784	4%	6804	**654**	0,27230	101	654	**0,00%**	0,52382	101	654	**0,00%**	0,07337	667	1,99%
200	784	4%	7864	**650**	0,22538	101	650	**0,00%**	0,48369	101	650	**0,00%**	0,07421	660	1,54%
200	784	4%	8883	**652**	0,29544	101	652	**0,00%**	0,15381	101	652	**0,00%**	0,07165	663	1,69%
200	1164	6%	2108	**1003**	1,26888	101	1003	**0,00%**	2,31309	101	1003	**0,00%**	0,14209	1022	1,89%
200	1164	6%	4085	**999**	1,84259	101	999	**0,00%**	1,89349	101	999	**0,00%**	0,13442	1018	1,90%
200	1164	6%	5984	**1005**	1,24284	101	1005	**0,00%**	1,22939	101	1005	**0,00%**	0,14184	1021	1,59%
200	1164	6%	6812	**1003**	1,65287	101	1003	**0,00%**	1,02836	101	1003	**0,00%**	0,14720	1019	1,60%
200	1164	6%	8199	**1003**	3,33555	101	1003	**0,00%**	2,73854	101	1003	**0,00%**	0,14033	1019	1,60%
200	1536	8%	6079	**1346**	6,08327	101	1346	**0,00%**	4,64867	101	1346	**0,00%**	0,22907	1371	1,86%
200	1536	8%	6083	**1340**	7,64292	101	1340	**0,00%**	6,51846	101	1340	**0,00%**	0,21920	1374	2,54%
200	1536	8%	6243	**1340**	11,28444	101	1340	**0,00%**	5,70559	101	1340	**0,00%**	0,20965	1370	2,24%
200	1536	8%	6341	**1347**	27,84275	101	1347	**0,00%**	19,08941	101	1347	**0,00%**	0,21606	1367	1,48%
200	1536	8%	8705	**1340**	11,07558	101	1340	**0,00%**	6,37973	101	1340	**0,00%**	0,21326	1365	1,87%
400	4656	6%	1547	**4244***	3600,01613	107	4244	**0,00%**	3600,02691	107	4249	0,12%	1,05018	4302	1,37%
400	4656	6%	1992	**4255***	3600,01522	107	4255	**0,00%**	3600,01416	107	4256	0,02%	1,07520	4311	1,32%
400	4656	6%	3708	**4245***	3600,02152	107	4248	0,07%	3600,00570	107	4245	**0,00%**	1,05052	4297	1,22%
400	4656	6%	3894	**4239***	3600,00365	107	4240	0,02%	3600,00433	107	4239	**0,00%**	1,04515	4304	1,53%
400	4656	6%	6033	**4244***	3600,00280	107	4245	0,02%	3600,02396	107	4244	**0,00%**	1,04795	4301	1,34%
400	6144	8%	2944	**5673***	3600,04475	107	5675	0,04%	3600,06056	107	5673	**0,00%**	1,66061	5746	1,29%
400	6144	8%	3004	**5672***	3600,06578	107	5672	**0,00%**	3600,00630	107	5686	0,25%	1,64912	5738	1,16%
400	6144	8%	6593	**5676***	3600,05217	107	5676	**0,00%**	3600,10692	107	5697	0,37%	1,66080	5749	1,29%
400	6144	8%	7816	**5663***	3600,04927	107	5663	**0,00%**	3600,00790	107	5669	0,11%	1,68958	5746	1,47%
400	6144	8%	9172	**5674***	3600,05297	107	5674	**0,00%**	3600,09185	107	5680	0,11%	1,67379	5756	1,45%
400	1584	2%	4537	**1340**	0,71252	101	1340	**0,00%**	0,62024	101	1340	**0,00%**	0,17874	1356	1,19%
400	1584	2%	4859	**1344**	0,69652	101	1344	**0,00%**	0,48717	101	1344	**0,00%**	0,20954	1361	1,26%
400	1584	2%	4905	**1331**	0,20476	101	1331	**0,00%**	0,16927	101	1331	**0,00%**	0,17694	1348	1,28%
400	1584	2%	6463	**1335**	0,38771	101	1335	**0,00%**	0,37347	101	1335	**0,00%**	0,18136	1350	1,12%
400	1584	2%	9979	**1335**	0,55882	101	1335	**0,00%**	0,43771	101	1335	**0,00%**	0,18565	1357	1,65%
400	3136	4%	4938	**2807**	59,38488	101	2807	**0,00%**	261,64278	101	2807	**0,00%**	0,57112	2843	1,28%
400	3136	4%	6094	**2798**	18,76451	101	2798	**0,00%**	15,50639	101	2798	**0,00%**	0,51734	2843	1,61%
400	3136	4%	8404	**2799**	18,44189	101	2799	**0,00%**	13,85526	101	2799	**0,00%**	0,51362	2835	1,29%
400	3136	4%	8716	**2796**	26,83602	101	2796	**0,00%**	18,65516	101	2796	**0,00%**	0,53707	2837	1,47%
400	3136	4%	9424	**2796**	31,01732	101	2796	**0,00%**	23,11187	101	2796	**0,00%**	0,54765	2833	1,32%

Table 4.1: Performance results on Barabási–Albert instances with $n \in \{100, 200, 400\}$.

the generation process. The fifth column (*Best*) reports, for each instance, the smallest known number of edges to be removed from the network in order to obtain a cluster graph.

The remaining columns of both the tables are grouped by the associated solution methods. With respect to Formulation 1 and Formulation 2, the tables report: the run time in seconds

					Formulation 1				Formulation 2				EC Heuristic		
Nodes	Edges	Density	Seed	Best	Time	Status	Value	Gap	Time	Status	Value	Gap	Time	Value	Gap
600	7056	4%	4368	**6486***	3600,02676	107	6486	**0,00%**	3600,02485	107	6487	0,02%	1,72893	6559	1,13%
600	7056	4%	6596	**6478***	3600,02499	107	6478	**0,00%**	3600,03310	107	6481	0,05%	1,77811	6569	1,40%
600	7056	4%	7686	**6483***	3600,03521	107	6483	**0,00%**	3600,03087	107	6486	0,05%	1,75355	6586	1,59%
600	7056	4%	8191	**6495***	3600,04858	107	6495	**0,00%**	3600,00466	107	6497	0,03%	1,78506	6573	1,20%
600	7056	4%	8728	**6484***	3600,03490	107	6486	0,03%	3600,03020	107	6484	**0,00%**	1,79210	6567	1,28%
600	10476	6%	2905	**9890***	3600,25301	107	9960	0,71%	3600,16089	107	9951	0,62%	3,44522	9890	**0,00%**
600	10476	6%	4340	**9890***	3600,23111	107	9942	0,53%	3600,01493	107	9935	0,46%	3,48582	9890	**0,00%**
600	10476	6%	6591	**9882***	3600,18233	107	9930	0,49%	3600,23875	107	9934	0,53%	3,51074	9882	**0,00%**
600	10476	6%	6698	**9886***	3600,09389	107	9938	0,53%	3600,21389	107	9972	0,87%	3,45336	9886	**0,00%**
600	10476	6%	6865	**9898***	3600,20801	107	9947	0,50%	3600,22602	107	9992	0,95%	3,51977	9898	**0,00%**
600	13824	8%	1598	**13148***	3690,53612	107	13824	5,14%	3676,62744	107	13824	5,14%	5,68697	13148	**0,00%**
600	13824	8%	4019	**13169***	3684,39407	107	13824	4,97%	3670,02102	107	13824	4,97%	5,63253	13169	**0,00%**
600	13824	8%	4399	**13167***	3600,26506	107	13324	1,19%	3672,06708	107	13824	4,99%	5,68386	13167	**0,00%**
600	13824	8%	4486	**13163***	3690,74541	107	13824	5,02%	3672,00909	107	13824	5,02%	5,62442	13163	**0,00%**
600	13824	8%	8841	**13156***	3600,32661	107	13313	1,19%	3676,05070	107	13824	5,08%	5,76818	13156	**0,00%**
600	3564	2%	3595	**3157**	10,84976	101	3157	**0,00%**	9,73925	101	3157	**0,00%**	0,55653	3194	1,17%
600	3564	2%	4888	**3142**	9,01697	101	3142	**0,00%**	6,18206	101	3142	**0,00%**	0,59686	3186	1,40%
600	3564	2%	7204	**3142**	9,01759	101	3142	**0,00%**	6,33457	101	3142	**0,00%**	0,53630	3180	1,21%
600	3564	2%	7629	**3147**	8,12684	101	3147	**0,00%**	6,10780	101	3147	**0,00%**	0,55879	3186	1,24%
600	3564	2%	9736	**3141**	4,79167	101	3141	**0,00%**	4,02185	101	3141	**0,00%**	0,53101	3184	1,37%
800	12544	4%	1036	**11864***	3600,17970	107	11922	0,49%	3600,16091	107	11887	0,19%	4,19278	11864	**0,00%**
800	12544	4%	1582	**11867***	3600,21947	107	11918	0,43%	3600,12744	107	11867	**0,00%**	4,13305	11869	0,02%
800	12544	4%	4704	**11852***	3600,20922	107	11917	0,55%	3600,12585	107	11976	1,05%	4,17858	11852	**0,00%**
800	12544	4%	8177	**11860***	3600,13144	107	11916	0,47%	3600,18231	107	11925	0,55%	4,12746	11860	**0,00%**
800	12544	4%	8746	**11865***	3600,11666	107	11932	0,56%	3600,18539	107	11950	0,72%	4,17969	11865	**0,00%**
800	18624	6%	4844	**17809***	3600,49858	107	17996	1,05%	3600,41282	107	18624	4,58%	8,43388	17809	**0,00%**
800	18624	6%	7892	**17784***	3600,48287	107	17995	1,19%	3600,38517	107	17993	1,18%	8,41124	17784	**0,00%**
800	18624	6%	8325	**17811***	3600,50613	107	18004	1,08%	3600,46701	107	18014	1,14%	8,26219	17811	**0,00%**
800	18624	6%	8965	**17822***	3600,54814	107	18034	1,19%	3600,34857	107	18172	1,96%	8,28237	17822	**0,00%**
800	18624	6%	8996	**17757***	3600,46816	107	17964	1,17%	3600,39491	107	17975	1,23%	8,26202	17757	**0,00%**
800	24576	8%	1562	**23640***	3600,66115	107	23839	0,84%	3601,34051	107	24576	3,96%	13,42387	23640	**0,00%**
800	24576	8%	1701	**23596***	3600,85330	107	24151	2,35%	3601,31008	107	24576	4,15%	13,42896	23596	**0,00%**
800	24576	8%	8183	**23629***	3600,77345	107	24159	2,24%	3600,07099	107	24576	4,01%	13,43441	23629	**0,00%**
800	24576	8%	9320	**23646***	3601,07822	107	23842	0,83%	3601,34101	107	24576	3,93%	13,33178	23646	**0,00%**
800	24576	8%	9837	**23680***	3602,55857	107	23861	0,76%	3601,22140	107	24576	3,78%	13,69588	23680	**0,00%**
800	6336	2%	2111	**5729**	87,87187	101	5729	**0,00%**	59,40477	101	5729	**0,00%**	1,32234	5800	1,24%
800	6336	2%	2123	**5733**	74,53751	101	5733	**0,00%**	49,89538	101	5733	**0,00%**	1,25029	5802	1,20%
800	6336	2%	5620	**5732**	92,89899	101	5732	**0,00%**	57,67675	101	5732	**0,00%**	1,25432	5801	1,20%
800	6336	2%	5909	**5734**	140,58885	101	5734	**0,00%**	124,08900	101	5734	**0,00%**	1,30032	5810	1,33%
800	6336	2%	7760	**5739**	1373,18027	101	5739	**0,00%**	1126,28710	101	5739	**0,00%**	1,24351	5798	1,03%
1000	9900	2%	1372	**9076**	816,22444	101	9076	**0,00%**	298,46308	101	9076	**0,00%**	2,40482	9190	1,26%
1000	9900	2%	1912	**9095**	3600,02067	107	9095	**0,00%**	481,23288	101	9095	**0,00%**	2,47602	9209	1,25%
1000	9900	2%	3522	**9136***	3600,19534	107	9147	0,12%	3600,08206	107	9136	**0,00%**	2,40871	9195	0,65%
1000	9900	2%	3672	**9098**	2291,69276	101	9098	**0,00%**	1645,71847	101	9098	**0,00%**	2,36529	9214	1,28%
1000	9900	2%	7114	**9127***	3600,11672	107	9154	0,30%	3600,16561	107	9127	**0,00%**	2,40562	9198	0,78%
1000	19600	4%	1233	**18693***	3600,61999	107	18864	0,91%	3600,32997	107	18895	1,08%	7,92907	18693	**0,00%**
1000	19600	4%	6400	**18669***	3600,77487	107	18886	1,16%	3600,42399	107	18871	1,08%	7,90791	18669	**0,00%**
1000	19600	4%	7392	**18694***	3600,80622	107	18903	1,12%	3600,41892	107	18902	1,11%	7,98831	18694	**0,00%**
1000	19600	4%	7696	**18689***	3600,52217	107	18902	1,14%	3600,42015	107	18934	1,31%	7,97740	18689	**0,00%**
1000	19600	4%	9570	**18717***	3600,43984	107	18897	0,96%	3600,43455	107	18920	1,08%	7,98703	18717	**0,00%**
1000	29100	6%	2867	**28039***	3600,83060	107	28296	0,92%	3600,89929	107	28319	1,00%	16,27523	28039	**0,00%**
1000	29100	6%	3177	**28038***	3600,91162	107	28582	1,94%	3600,85727	107	28524	1,73%	16,15439	28038	**0,00%**
1000	29100	6%	3708	**28037***	3600,99056	107	28243	0,73%	3601,24830	107	28293	0,91%	16,42042	28037	**0,00%**
1000	29100	6%	4441	**28008***	3601,49785	107	28585	2,06%	3601,14618	107	28260	0,90%	16,17299	28008	**0,00%**
1000	29100	6%	7918	**28015***	3601,09408	107	28260	0,87%	3601,06337	107	28307	1,04%	16,25776	28015	**0,00%**
1000	38400	8%	3819	**37202***	3601,57106	107	37882	1,83%	3600,16216	107	37535	0,90%	26,94874	37202	**0,00%**
1000	38400	8%	4940	**37191***	3602,59315	107	37869	1,82%	3600,12356	107	37510	0,86%	26,71472	37191	**0,00%**
1000	38400	8%	6235	**37173***	3601,95774	107	38400	3,30%	3600,22133	107	37409	0,63%	26,57302	37173	**0,00%**
1000	38400	8%	8397	**37196***	3602,49115	107	37420	0,60%	3603,35665	107	38400	3,24%	26,76940	37196	**0,00%**
1000	38400	8%	8632	**37163***	3601,07574	107	37448	0,77%	3600,27340	107	37489	0,88%	26,63642	37163	**0,00%**

Table 4.2: Performance results on Barabási–Albert instances with $n \in \{600, 800, 1000\}$.

spent by the solver to obtain such solution (*Time*); the status code reported by CPLEX at the end of the computation (*Status*), which mainly equals to 101 when the problem has been solved to optimality, to 107 when the given time limit has been reached after computing a feasible solution and to 108 when no feasible solution has been produced before reaching the time limit;

the value associated with the returned solution (*Value*); and finally the relative percentage gap of the returned solution with respect to the best known one (*Gap*). Finally, the *Time*, *Value* and *Gap* columns are also reported for the edge contraction heuristic, with analogous meanings to those of the exact methods. Note that, by construction, the value of the solution provided by the heuristic also represents the actual number of iterations carried out by the algorithm.

Although the objective functions (4.1a) and (4.2a) of both the formulations express the number of edges belonging to the resulting cluster graph, for the sake of readability we report, in the *Value* columns, its complement with respect to the total number of edges, i.e. the number of edges to be removed. By doing so, the *Value* columns of the two formulations are directly comparable to the one associated with the heuristic. Consequently, the relative gap of each method is computed as $100\% \cdot (Value - Best)/Best$. Such gap is zero when the related method finds a solution whose value is lower or equal to the ones provided by the other methods. If this is the case, the gap is reported in bold style. Clearly, on a given instance, at least one method has 0.00% gap and it is possible that more than a single method find equivalent or identical solutions whose value is the best known one.

Note that the best known solution value necessarily coincides with the value of the solution provided by one of the two formulations if such formulation completed the computation by returning the optimal solution. On the other hand, in case the execution of both the formulations led to early termination due to time limit budget, the best known solution value is the one associated to the best known feasible solution, i.e., the minimum one among the values of the three solutions returned by the two formulations and the heuristic. Since, in the latter case, the optimality of the best known solution is not certified, an asterisk is added alongside such value in the *Best* column.

Table 4.3 summarizes the quality and time performances of the proposed methods on the addressed Barabási-Albert networks, by aggregating the results associated with instances belonging to the same scenario. As a result, the table contains 24 rows, each one reporting the average values computed with respect to the instances of the scenario identified by a given combination of the number of nodes, the number of edges and the density of the network. For each method, the required computation times and the relative gaps of the returned solutions are reported. Additionally, the overall average time and gap values, as well as the number of best solution identified by each method, are shown in the very last rows of the table.

According to the average computation time, Formulations 1 and 2 are almost equivalent. The former exhibits a slightly better relative gap value (0.47% versus 0.67%) and identifies two more best solution than the latter (73 versus 71). However, Tables 4.1 and 4.2 show that the subsets of instances on which each formulation finds the best solutions do not perfectly overlap and the total number of instances for which at least one of them identifies the best solution is 81. Among these, 63 is the number of certified optimal solutions.

On the other hand, the heuristic identifies the best known solution on a subset of 48 instances. In particular, in 9 cases, the returned solution is equivalent to the optimal one returned by one or both the exact formulations, while for the remaining 39 instances the heuristic produces a solution whose value is better than the one obtained with the exact methods. The performance

				Formulation 1		Formulation 2		EC Heuristic	
Nodes	**Edges**	**Density**	**Best**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**
100	99	2%	64,8	0,00193	**0,00%**	0,00179	**0,00%**	0,00666	**0,00%**
100	196	4%	149,2	0,02404	**0,00%**	0,01117	**0,00%**	0,01380	0,40%
100	291	6%	230	0,02500	**0,00%**	0,02705	**0,00%**	0,02677	1,31%
100	384	8%	311,2	0,05679	**0,00%**	0,09184	**0,00%**	0,03259	1,54%
200	396	2%	305,2	0,02366	**0,00%**	0,02081	**0,00%**	0,03102	0,39%
200	784	4%	651	0,23665	**0,00%**	0,29661	**0,00%**	0,07483	1,66%
200	1164	6%	1002,6	1,86855	**0,00%**	1,84057	**0,00%**	0,14118	1,72%
200	1536	8%	1342,6	12,7858	**0,00%**	8,46837	**0,00%**	0,21745	2,00%
400	4656	6%	4245,4	3.600,01	0,02%	3.600,02	0,03%	1,05380	1,36%
400	6144	8%	5671,6	3.600,05	0,01%	3.600,05	0,17%	1,66678	1,33%
400	1584	2%	1337	0,51207	**0,00%**	0,41757	**0,00%**	0,18644	1,30%
400	3136	4%	2799,2	30,8889	**0,00%**	66,5543	**0,00%**	0,53736	1,39%
600	7056	4%	6485,2	3.600,03	0,01%	3.600,02	0,03%	1,76755	1,32%
600	10476	6%	9889,2	3.600,19	0,55%	3.600,17	0,68%	3,48298	**0,00%**
600	13824	8%	13160,6	3.653,25	3,50%	3.673,36	5,04%	5,67919	**0,00%**
600	3564	2%	3145,8	8,36057	**0,00%**	6,47711	**0,00%**	0,55590	1,28%
800	12544	4%	11861,6	3.600,17	0,50%	3.600,16	0,50%	4,16231	**0,00%**
800	18624	6%	17796,6	3.600,50	1,14%	3.600,40	2,02%	8,33034	**0,00%**
800	24576	8%	23638,2	3.601,18	1,41%	3.601,06	3,97%	13,4630	**0,00%**
800	6336	2%	5733,4	353,8155	**0,00%**	283,4706	**0,00%**	1,27415	1,20%
1000	9900	2%	9106,4	2.781,65	0,08%	1.925,13	**0,00%**	2,41209	1,04%
1000	19600	4%	18692,4	3.600,63	1,06%	3.600,41	1,13%	7,95794	**0,00%**
1000	29100	6%	28027,4	3.601,06	1,31%	3.601,04	1,12%	16,2562	**0,00%**
1000	38400	8%	37185	3.601,94	1,66%	3.600,83	1,30%	26,7285	**0,00%**
			Avg	1785,3870	0,47%	1748,7633	0,67%	4,0024	0,80%
			#Best	73		71		48	

Table 4.3: Aggregated quality and time performances of the two exact and one heuristic proposed approaches on the 24 Barabási–Albert instances scenarios.

is significantly promising, since the overall relative gap collected by the heuristic algorithm is 0.80%, with a peak of 2.54%, while the average computation time is ∼4 seconds and the algorithm never requires more than 27 seconds to solve a single instance of the CD problem.

In Figure 4.1, we investigate how the number of acceptable solutions found by each method increases with respect to both a maximum computation time and an adjustable threshold τ on the solution quality. We consider $\tau \in \{0, 0.01, 0.02, 0.03\}$, meaning that, in order to be taken into account, the generated solutions must have relative gap values at most equal to $0\%, 1\%, 2\%$ and 3% from the best known ones, respectively. As we can see, when $\tau \leq 0.01$, i.e. in the two graphics above, the exact formulations are more effective, since they solve, in one hour of computation, a larger number of instances according to the quality requirements. The situation completely changes when we consider $\tau = 0.02$ in the left below graphic: in this case, the convenience of using the heuristic algorithm is clear, since it is able to solve almost the whole set of instances according to the specified quality criterion within ∼28 seconds. The described behavior is even more evident when considering $\tau = 0.03$ in the right below graphic: indeed, this threshold value is enough for the heuristic algorithm to solve the whole set of 120 instances always identifying an acceptable solution; the two exact formulations, instead, solve a few less instances, since there

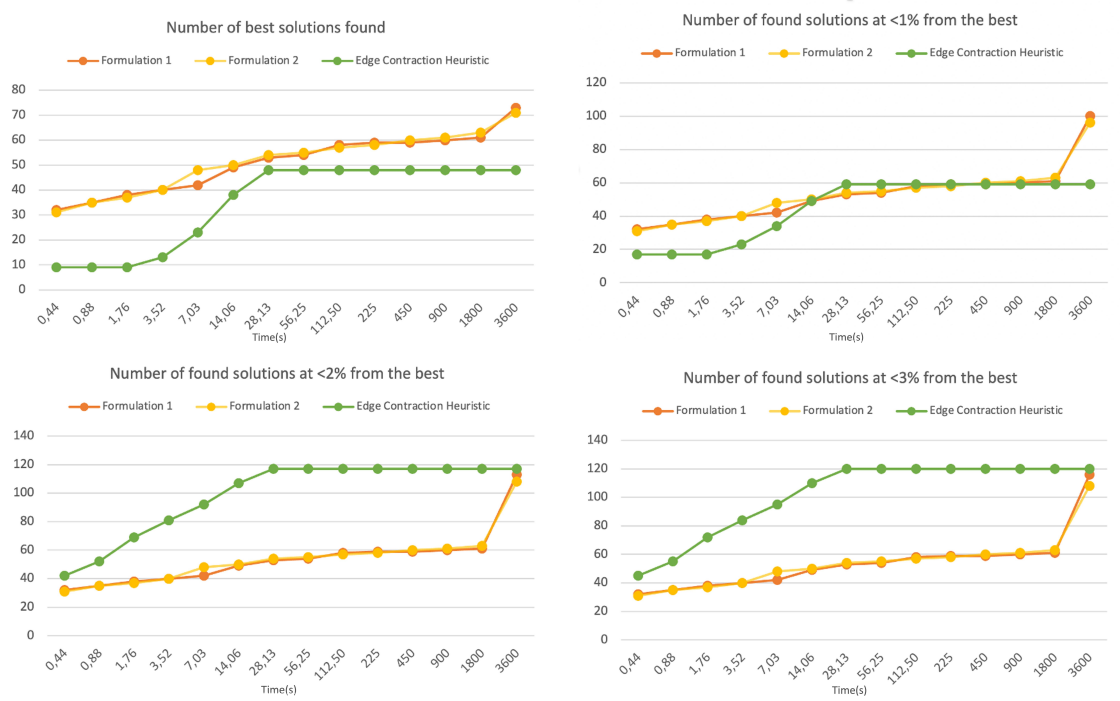


Figure 4.1: Number of solutions found by Formulation 1, Formulation 2 and `ECHeuristic` having values lower or equal to $(1+\tau)$ times the best known one, $\tau \in \{0, 0.01, 0.02, 0.03\}$.

exist instances for which the heuristic provides solutions which are at least 3% more convenient than the solutions identified by the exact formulations.

In Figure 4.2, we further analyze how the number of the best solutions identified by each of the proposed methods changes as the number of nodes (left) and the network density (right) increase. As previously described, we have six different values of $n \in \{100, 200, 400, 600, 800, 1\,000\}$, each of which is associated with a set of 20 instances, and four different values of $d \in \{2\%, 4\%, 6\%, 8\%\}$, resulting in four sets of 30 instances each. As one may expect, the higher the values of n and d, the larger the number of best solutions identified by `ECHeuristic`. In particular, the algorithm finds a larger number of best solutions than both the exact formulations for $n \geq 400$ and $d \geq 6\%$.

Finally, Figure 4.3 shows the clear convenience of the heuristic method with respect to the time required for solving an instance of the CD problem. Computational times are reported, in log-scale, on the y-axis of the graphic, while the values on the x-axis are the number of nodes of the several subsets of instances. Because of the time limit of one hour imposed for the exact formulations, the related curves never go over $log(3\,600)$. Nevertheless, the increase in the required time, with respect to the increase in the number of nodes in the network, is evidently more pronounced for the two ILP formulations than for the heuristic algorithm.

4.4.2 Biological networks

Biological networks were collected from Network Repository [RA15], which provides a large selection of real-world scientific network datasets from several domains. More in detail, a set of 28 biological networks was addressed in our experiments, with number of nodes $n \in [453, 15\,229]$, number of edges $m \in [1\,129, 245\,952]$ and density $d \in [0.07\%, 2.90\%]$.

				Formulation 1		Formulation 2		EC Heuristic			Relative Gap		
Instance	Nodes	Edges	Density	Value	Time	Value	Time	Value	Time	Best	Form. 1	Formu. 2	EC Heur.
bio-DM-LC	658	1129	0,52%	865	0,06	865	0,07	870	0,06	**865**	**0,00%**	**0,00%**	0,58%
bio-diseasome	516	1188	0,89%	495	0,39	495	0,33	496	0,09	**495**	**0,00%**	**0,00%**	0,20%
bio-grid-mouse	1450	1636	0,16%	1171	0,06	1171	0,07	1173	0,14	**1171**	**0,00%**	**0,00%**	0,17%
bio-CE-LC	1387	1648	0,17%	1197	0,07	1197	0,08	1202	0,08	**1197**	**0,00%**	**0,00%**	0,42%
bio-yeast	1458	1948	0,18%	1413	0,14	1413	0,10	1422	0,12	**1413**	**0,00%**	**0,00%**	0,64%
bio-celegans	453	2025	1,98%	1634	152,76	1634	145,47	1669	0,45	**1634**	**0,00%**	**0,00%**	2,14%
bio-CE-HT	2617	2985	0,09%	2150	0,06	2150	0,06	2154	0,21	**2150**	**0,00%**	**0,00%**	0,19%
bio-grid-plant	1717	3098	0,21%	2269	0,34	2269	0,45	2300	0,31	**2269**	**0,00%**	**0,00%**	1,37%
bio-CE-GT	924	3239	0,76%	2589	3,95	2589	3,21	2618	0,48	**2589**	**0,00%**	**0,00%**	1,12%
bio-SC-TS	636	3959	1,96%	1217	6,21	1217	9,66	1228	0,43	**1217**	**0,00%**	**0,00%**	0,90%
bio-DM-HT	2989	4660	0,10%	3634	0,13	3634	0,16	3642	0,31	**3634**	**0,00%**	**0,00%**	0,22%
bio-grid-worm	3507	6531	0,11%	5740	5,38	5740	5,91	5759	2,10	**5740**	**0,00%**	**0,00%**	0,33%
bio-grid-fission-yeast	2026	12637	0,62%	11619	3600,48	11746	3600,26	11484	6,75	**11484***	1,18%	2,28%	**0,00%**
bio-HS-HT	2570	13691	0,41%	10621	565,18	10621	391,19	10874	2,52	**10621**	**0,00%**	**0,00%**	2,38%
bio-SC-LC	2004	20452	1,02%	20428	3659,41	18229	3600,22	18213	5,31	**18213***	12,16%	0,09%	**0,00%**
bio-grid-fruitfly	7274	24894	0,09%	21823	44,89	21823	29,68	21972	7,45	**21823**	**0,00%**	**0,00%**	0,68%
bio-dmela	7393	25569	0,09%	22754	8,79	22754	9,07	22873	4,74	**22754**	**0,00%**	**0,00%**	0,52%
bio-grid-human	9436	31182	0,07%	26723	2737,98	26971	3600,31	27010	11,12	**26723**	**0,00%**	0,93%	1,07%
bio-SC-GT	1716	33987	2,31%	30651	3601,94	31323	3601,35	30488	15,33	**30488***	0,53%	2,74%	**0,00%**
bio-SC-CC	2223	34879	1,41%	30146	3601,68	30127	3602,57	29510	17,31	**29510***	2,16%	2,09%	**0,00%**
bio-HS-LC	4227	39484	0,44%	36080	3601,08	37727	3600,96	34168	14,40	**34168***	5,60%	10,42%	**0,00%**
bio-CE-PG	1871	47754	2,73%	46073	3600,54	45939	3600,14	46061	39,51	**45939***	0,29%	**0,00%**	0,27%
bio-CE-GN	2220	53683	2,18%	51597	3602,72	51712	3600,19	51146	31,40	**51146***	0,88%	1,11%	**0,00%**
bio-SC-HT	2084	63027	2,90%	58438	3600,83	63020	3606,33	56767	53,55	**56767***	2,94%	11,02%	**0,00%**
bio-DM-CX	4040	76717	0,94%	70532	3600,89	76700	3651,44	68668	41,88	**68668***	2,71%	11,70%	**0,00%**
bio-DR-CX	3289	84940	1,57%	81424	3601,43	84939	3606,75	77690	62,97	**77690***	4,81%	9,33%	**0,00%**
bio-HS-CX	4413	108818	1,12%	102503	3601,45	108810	3611,24	98681	77,61	**98681***	3,87%	10,26%	**0,00%**
bio-CE-CX	15229	245952	0,21%	245735	3676,82	245735	3670,54	221228	150,98	**221228***	11,08%	11,08%	**0,00%**
					1674,13		1698,14		19,56		1,72%	2,61%	0,47%

Table 4.4: Comparison of the performances of the proposed approaches on real biological networks.

Table 4.4 reports the results obtained by running the three proposed solution approaches

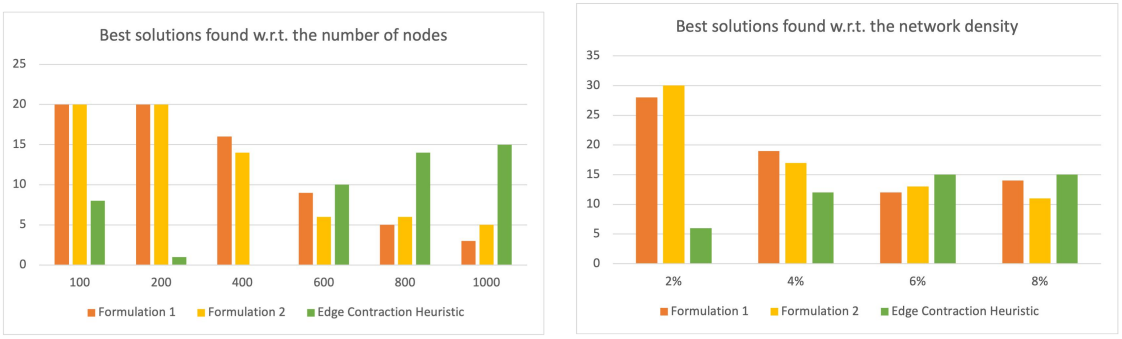


Figure 4.2: Number of best solutions found by Formulation 1, Formulation 2 and `ECHeuristic` with respect to the number of nodes (on the left) and the network density (on the right).

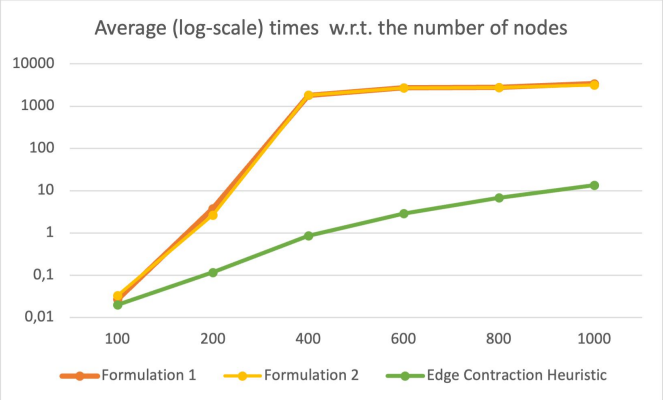


Figure 4.3: Increasing trends of the running times of Formulation 1, Formulation 2 and `ECHeuristic` as the number of nodes increases.

on the set of biological networks, sorted by increasing number of edges. For each method, the value of the returned solution, in terms of number of edges to be removed, as well as the total computation time in seconds, are reported. As before, the *Best* column shows which is the value of the best solution found and the subsequent columns report the relative gaps with respect to such value.

The experiments emphasize the efficiency and effectiveness of the proposed heuristic, which with an average computation time of ∼20 seconds and peak of ∼150 seconds on the larger instance, exhibits a relative gap of 0.47%, lower than the ones associated to both the exact formulations, equal to 1.72% and 2.61%, respectively. Furthermore, the algorithm finds the best known solution on 11 instances of this set, never exceeding the 2.38% gap in the remaining cases.

As for the two exact formulations, which together solve 16 instances to optimality, these tests show that when the size of the network significantly increases, Formulation 2 becomes less effective than Formulation 1, which, excluding a very restricted subset of instances, results slightly faster and produces considerably better solutions.

Part II

Combinatorial optimization problems with conflict constraints

Chapter 5

The Set Covering problem with Conflicts on Sets

The first computational optimization problem with conflict constraints analyzed in this thesis is the Set Cover problem with Conflicts on Sets (SCCS), it represents a particular extension of the classic Set Cover problem. Unlike the other problems with conflict treated in this thesis, the starting problem (without conflict constraints), in this case the classic Set Cover problem (SCP) is already NP-Hard. We study an extension of the SCP that is better suited to some real-world problems, introducing the concept of conflict not in a hard way as it is already used in the literature, but allowing the use of conflicts when necessary, paying a penalty.
The aim of the problem is to find the best combination of subsets to represent a cover of minimum cost, trying to reduce the cost of conflicts as well. We present two mathematical formulations for the SCCS problem and a parallel GRASP approach, which solves it heuristically. The typical structure of a GRASP algorithm is described in Section A.3.1 of Appendix A. A large set of instances were used to test our approach. The instances were created from instances already present in the literature, which were built to be challenging for the classic set cover proposed. The proposed heuristic is far more effective and efficient than Gurobi solver, according to computational tests, which solves the mathematical formulations in a much longer time.

Given a finite set of elements (items) $U = \{1, \ldots, m\}$ and $\{ U_j \subseteq U \mid j \in N \}$ be a collection of subsets of U where $N = \{ 1, \ldots, n \}$. With each U_j, $j \in N$, a non-negative cost c_j is associated. We say that $W \subseteq N$ covers U if $\bigcup_{j \in W} U_j = U$. The Set Covering Problem (SCP) asks for the cheapest cover. A bibliography of applications can be found in the appendix of [BP76] and in the book edited by [DMM97]. The problem is well known to be NP-hard and has been extensively studied in a variety of application contexts, including manufacturing and crew scheduling problems in railways to location. Also a variety of solution approaches, both exact and heuristic, have been proposed for the problem (Caprara et al. [CTF00]).

Another version instead, if an element remains uncovered a penalty has to be paid, as it happens in the prize-collecting version of partial SCP.

As for other combinatorial problems, a relevant role is played in the literature by variants. In recent years several variants has been studied including Set Covering generalizations or the introduction of additional features such as profits, budget constraint or conflicts. In the partial SCP it is either not necessary or not possible to cover all of the elements of the set U due to other constraints or objectives. Each element $i \in U$ is associated with a profit p_i, whereas each subset U_j has a cost. The problem aims to find a minimum cost collection of subsets such that the combined profit of the elements covered by the collection is at least equal to a predefined profit bound. In the prize-collecting version of partial SCP, elements should not be all strictly covered; however, if an element remains uncovered a penalty has to be paid. The problem looks for a collection of subsets such that the cost of selected subsets plus the penalties of uncovered elements is minimized (see Könemann et al. [KPS11] and references therein). In Bilal et al. [BGG14], the authors analyze a variant of the partial SCP where subsets are partitioned into groups. The selection of a subset in a solution implies the activation of its group and the payment of the cost associated with that group (negative profit). The objective is to maximize the total profit while it is not necessary to cover all the elements.
Here, we tackle a different problem called Set Covering with Conflicts on Sets (SCCS), where the meaning of conflict is determined by the total number of elements that any given pair of subsets cover together. If the number of elements in common between two sets exceeds the k threshold, then they are in conflict; these can still be selected by paying a penalty. This penalty is determined by how much the intersection cardinality of the subsets exceeds a certain threshold k. An interesting application of the SCCS concerns the location of radio antennas, (radio base station RBS), for mobile communication, taking into account not only the coverage through the signal of a certain area, but also the radioactive pollution that these antennas generate.

The growth in the number of Internet users and applications that make massive use of networked data exchange, such as online games, video streaming, telemedicine, lessons via videoconferencing and the Internet of Things (IoT), has led to increasing connectivity between devices of different types. Future communication networks will have to be implemented in order to create an infrastructure capable of simultaneously supporting various services and a large number of users connected to the service [ALL11],[GWC+13].
To support users and ensure high performance in densely populated urban areas, many small cells need to be located to provide good coverage. Nowadays people are more attentive and sensitive to the problem of possible dangers due to electromagnetic pollution generated by the use of these antennas [STM17].
Although no scientific evidence has been demonstrated of any relationship between electromagnetic fields and human disease, all precautionary measures against an uncontrolled rise in the electromagnetic field level must be taken. This choice is not only linked to a general precautionary principle, but also to specific legislation: in fact, the Italian regulations, in the field of electromagnetic emissions, impose a limit of 6 V/m for long-term exposed persons, and affirm that, however, the e.m. emissions in the field must be reduced to a minimum compatibly with the quality of services [CDLMR02]. In recent years the interest in the reduction of electromagnetic pollution has also spread to the scientific community of optimization; here are some articles that

deal with this topic [ACMS02],[WJG13].
Having also ensured a good quality of service (QoS), which to many may seem contrary to the precautionary principle, we want to cover all areas of interest avoiding installing the antennas (RBS) too close to each other in order to reduce the radiation exposure of people living near these antennas.
Suppose we want to cover a large city, the places where antennas can be placed are limited by physical and natural limits but they are not necessarily few. Considering the signal coverage of an antenna, we identify a specific region served by an RBS, which will have intersection points that will not only benefit from better signal reception but also from greater electromagnetic exposure. We could choose a subset of areas avoiding overlaps and minimizing radiation exposure, however, this leads to a reduction in the quality of the service, there may be coverage holes.
So our goal is to define a coverage of the city by allowing the intersections of the areas but paying a penalty proportional to the size of the intersection, so we want to add a new RBS only when it is strictly necessary.
In order to deal, with the coverage of the entire city but also with the electromagnetic pollution, we introduce a threshold k, which defines the maximum number of elements that two areas can have in common without paying a penalty.
The introduction of this threshold allows not only to reducing the intersection areas but also to ensure a good quality of service, having a strong impact on traffic and the quality of the connection. A configuration that completely eliminates intersections does not necessarily guarantee good coverage, so it is possible to modify this threshold, allowing the design of energy efficient systems with controlled levels of electromagnetic pollution, without compromising the quality of the experience of the user.

5.1 Mathematical Models

Given a finite set of elements $U = \{1, \dots, m\}$ and a collection of subsets $\{U_j \subseteq U \mid j \in N\}$, where $N = \{1, \dots, n\}$ contains the indices of the subsets of U. A non-negative cost c_j is associated with each subse, $j \in N$. Two subsets U_j and U_l are *in conflict* when they have more than k elements in common, i.e., $|U_j \cap U_l| > k$, where k is an integer called the *conflict threshold*. We define B as the set of all unordered pairs $\{j, l\}$, $j, l \in N$, $j \neq l$ of subsets and by $D \subseteq B$ the set of pairs $\{j, l\}$ such that the subsets U_j and U_l are in conflict. A non-negative conflict cost d_{jl} must be paid each time a conflict arises between two sets U_j and U_l in order to choose both of them as part of the solution. We set $d_{jl} = \max\{|U_j \cap U_l| - k, 0\}$, and we multiply it by γ which is a unitary cost:

$$\gamma := \max\left\{ \left\lceil \max_{i \in U} \frac{c_i}{|U_i|} \right\rceil, 1 \right\}.$$

To avoid the cost of conflicts being overlooked by the cost of coverage, we multiply the cost of conflicts by γ. Where γ is chosen in an adaptive way for each instance, setting it to the maximum

of the subset cost ratios for covered items. That is, it grows in relation to the cost of coverage. A cover of U that minimizes the sum of covering and conflict costs is aimed by the Set Covering Problem with Conflict on Sets (SCCS).

5.1.1 Binary Linear Programming Formulation

A pure binary model with two sets of binary variables is the simplest formulation. The first set of variables, is represented by the x_j variables, which is one for each sub-set U_j,

$$x_j = \begin{cases} 1 & if \quad U_j \; is \; selected \\ 0 & otherwise \end{cases}$$

The second set of variables is represented by y_{jl} $(j < l)$ for each possible pair $\{ j, l \} \in D$ of subsets in conflict.

$$y_{jl} = \begin{cases} 1 & if \; both \; U_j \; and \; U_l \; are \; selected \\ 0 & otherwise \end{cases}$$

The variable y_{jl} takes value 1 when both subsets are selected and zero otherwise. In order to define the integer linear programming model, we make use of a matrix A which has the elements u_i on the rows and the subsets U_j on the columns:

$$a_{ij} = \begin{cases} 1 & if \quad u_i \in U_j \\ 0 & otherwise \end{cases}$$

The mathematical formulation is as follows:

$$(\mathbf{SCCS\_BIN}) \quad \min \sum_{j \in N} c_j x_j + \sum_{\{ j,l \} \in D} d_{jl} y_{jl} \tag{5.1}$$

$$\sum_{j \in N} a_{ij} x_j \geq 1, \quad i \in U, \tag{5.2}$$

$$x_j + x_l \leq y_{jl} + 1, \quad \{ j, l \} \in D, j < l, \tag{5.3}$$

$$x_j \in \{ 0, 1 \}, \quad j \in N, \tag{5.4}$$

$$y_{jl} \in \{ 0, 1 \}, \quad \{ j, l \} \in D, j < l. \tag{5.5}$$

Constraints (5.2) are classical set covering constraints and constraints (5.3) model the conflict: when both subsets U_j and U_l are used in the solution, variable y_{jl} is forced to 1, it means $\{ j, l \} \in D$ are selected, i.e., $x_j = x_l = 1$. This model has $O(n^2)$ binary variables since the size of D is bounded by $O(n^2)$, and $O(n^2 + m)$ constraints.

5.1.2 Integer Linear Programming Formulation

The previous model (SSCS_BIN) can be reformulated using binary and integer variables. We define an integer variable w_{ij} defined for each pair of subsets U_j and U_l such that $\{ j, l \} \in B$, equal to the number of elements jointly belonging to both subsets if larger than threshold k and zero in all the other cases, thanks to objective function minimization.

$$(\textbf{SCCS\_MIP}) \quad \min \sum_{j \in N} c_j x_j + \gamma \sum_{\{ j,l \} \in B} w_{jl} \tag{5.6}$$

$$\sum_{j \in N} a_{ij} x_j \geq 1, \quad i \in U, \tag{5.7}$$

$$\sum_{i=1}^{m} a_{ij} a_{il} (x_j + x_l - 1) - k \leq w_{jl}, \quad \{ j, l \} \in B \tag{5.8}$$

$$x_j \in \{ 0, 1 \}, \quad j \in N, \tag{5.9}$$

$$w_{jl} \geq 0 \text{ integer}, \quad \{ j, l \} \in B. \tag{5.10}$$

Constraints (5.8) model the conflicts, so model the integer variable w_{jl} setting it to $\max\{|U_j \cap U_l| - k, 0\}$ if both sets U_j and U_l are selected and zero otherwise. Since the number of integer variables grows as the square of the number of available subsets n, the model consists of $O(n^2)$ integer variables, $O(n)$ binary variables and $O(n^2 + m)$ constraints.

5.2 The parallel GRASP algorithm

In this section, we describe an algorithm based on the GRASP (Greedy Randomized Adaptive Search Procedure) strategy that we implemented to solve the SCCS problem. It is a two-step procedure: first, an initial feasible solution is discovered using a combination of random and greedy selections, and then, in the second phase, a local search is used to enhance the solution found in the first step. We want to use a parallel implementation of the GRASP algorithm where common information is shared among different processes, each of which is performing the identical single-process GRASP. The parallel execution allows for a much broader range of the search space to be explored within the same time constraint, resulting in a significant improvement in the final solution, allowing to make the most of the hardware of the machine on which the tests are performed.
The single-process GRASP is first presented in the next section, after which its parallel version is introduced and we offer a data structure analysis with space and temporal complexity.

5.2.1 Single process GRASP

As for a regular GRASP there are two main phases to the method:

1. a feasible solution is built starting from an empty one by adding a subset U_j, $j \in N$ at a time.

2. A local search is conducted in the second stage, which involves sequentially exploring three different neighborhoods and involves a growing computational burden.

The first phase works in the following way. In order to select the next subset to add, all subsets are ranked using a greedy function that measures the benefit of each subset selection, obviously it is only an indicative value. Precisely the various sub-sets U_j are ordered in non-decreasing order of the ratio of their costs plus the conflicts costs with the subsets that already belong to the current partial solution (it is initially empty, so the first choice does not include any conflict costs), over their cardinality excluding the items already covered by the current solution (we refer to this list as *Candidate List*). The heuristic is adaptive because the benefits related to each subset are updated based on the subsets already chosen in the partial solution at each iteration. A candidate is selected at random from *Restricted Candidate List* (RCL) that contains a predetermined number of the most promising subsets. As a result, various solutions might be obtained at the conclusion of each GRASP iteration. The local search in the second phase, as already mentioned, uses three types of moves, which allows to explore three types of neighborhoods, these are respectively one more expensive than the other. The algorithm stops using the first neighborhood only when it remains blocked in a local minima, so it can no longer improve, and move on to the second neighborhood, in the same way when the second neighborhood fails to explore more new solutions, it moves to the third.
In writing the pseudo-code, we used the following conventions:

- For simple numerical values, we adopted lower case letters (a, b, c, ...);
- For array containing simple numerical values, upper case letters (A, B, C, ...);
- For the associative arrays containing sets of numerical values we used upper case letters with calligraphic font ($\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$, ...).

Te value associated to a given key-index k is denoted placing k as subscript to the name of the data structure, in both cases: (A_k, B_k, C_k, ..., $\mathcal{A}_k$, $\mathcal{B}_k$, $\mathcal{C}_k$, ...).

Algorithm 5: Single-process-GRASP

Data: $instance, i_{\max}, t_{\max}, S, sharedCache, incumbentCost$
Result: Incumbent solution W^* and its objective value w^*

```
1  I ← BuildIncidenceStructure(instance) ;
2  C ← BuildConflictStructure(instance) ;
3  (W*, w*) ← (N, +∞) ;
4  while (i < i_max) and (t < t_max) do
5  |  (W, w) ← (∅, 0) ;
6  |  (R, V, Q) ← InitStateStructures(instance) ;
7  |  GraspPhase1(instance, C, I, W, w, R, V, Q, S, sharedCache) ;
8  |  GraspPhase2(instance, C, I, W, w, R, V, Q) ;
9  |  if w < w* then
10 |  |  (W*, w*) ← (W, w);
11 |  |  i ← 0;
12 |  else
13 |  |  i ← i + 1;
14 |  end
15 |  t ← ElapsedTime()
16 end
17 return (W*, w*)
```

Algorithm 5 describe the procedure behind our parallel algorithm. The algorithm takes as input:

- *instance*: the data related to a problem, (i.e., the set of elements U, all subsets U_j, $j \in N$, the cost of each subset and the non-zero conflict costs);
- $i_{\max}$,indicates the maximum number of iterations without improvement after which the algorithm terminates (the stopping rule parameters).
- $t_{\max}$ is the time limit assigned to the algorithm;
- S, $S \subseteq N$ refers to the family of subsets selected to build the initial cover during the first phase of GRASP (at the beginning of execution S is empty);
- *sharedCache* is the data structure that contains common information that is shared by all processes;
- *incumbentCost* is the objective function value of the best solution found among all processes at any time (if we run the parallel variant of the algorithm).

Note that, the data used by the parallel algorithm variant is already included in our description even though the algorithm run in a single process. Since this information is only relevant if the algorithm is executed in parallel, we assume that $S = N$.

Lines 1–3 of the Algorithm 5 are initialization steps. `BuildIncidenceStructure` is a function that builds and returns, the array $\mathcal{I}$ that associates with each element $i \in U$ the subsets U_j containing i, i.e. $\mathcal{I}_i \coloneqq \{ j \in N \mid i \in U_j \}$. This set can be immediately used to check if the input instance is infeasible, checking if any subset $\mathcal{I}_i$, $i \in U$ is empty. Similarly the function `BuildConflictStructure` builds the associative array $\mathcal{C}$ that maps each index $j \in N$ to the set C_j, and we define it in the following way: $C_j = \{ l \in N \mid d_{jl} > 0 \}$, $j \in N$ the set of those subsets of U that are in conflict with U_j. At Line 3, the incumbent solution W^* and its objective function value w^* are initialized.

Algorithm 5 include a main cycle (Lines 4–16) which stops as soon as one of the two stopping rules, controlled by $i_{\max}$ and $t_{\max}$, holds. During each iteration we initialize an empty set W that will contain the indexes of the cover we are going to build and we set its initial cost w equal to 0. At line 6 we use the procedure `InitStateStructures`, which initializes structures which are: $R, \mathcal{V}, \mathcal{Q}$. Let's define $R_j = c_j + \sum_{l \in W} d_{jl}$ to represent the *current cost* for each subset U_j, $j \in N$, corresponding to the partial cover W. We say that R_j represents the increase of the objective function value if j were added to the current working cover W, only if $j \notin W$, then. While if $j \in W$, then R_j represents the decrease of the objective function value if j were removed from the current working cover W. In order to implement this mechanism, we make use of the associative array R. Since W is empty at the start of each cycle, the current cost of each subset U_j equals its subset cost, i.e. $R_j = c_j$ for each $j \in N$; thus, array R is initialized with the instance initial costs C. As for the structure $\mathcal{V}$, we denote by $\mathcal{V}_i \coloneqq \{ j \in W \mid i \in U_j \}$ for each element $i \in U$, the set of subsets U_j, $j \in W$ that cover i. Similarly, the $\mathcal{Q}$ structure is thus defined: by $\mathcal{Q}_j = \{ i \in U \mid \mathcal{V}_i = \{ j \} \}$ the set of elements of U uniquely covered by subset U_j for a given $j \in W$. Data structures $\mathcal{V}$ and $\mathcal{Q}$ are implemented as associative arrays mapping each element $i \in U$ to $\mathcal{V}_i$ and each subset U_j, $j \in W$ to $\mathcal{Q}_j$. Furthermore, because they rely on W, they are updated anytime W changes and since W is initially empty, the data structures $\mathcal{V}$ and $\mathcal{Q}$ are initially empty as well.
The two phases of our algorithm, are reported in lines 7-8, using the two functions `GraspPhase1` and `GraspPhase2`. The first function tries to populate the working cover W in order to get an initial, possibly good solution. The latter tries to improve the initial solution exploring a sequence of neighbourhoods. At lines 9–14 the algorithm checks if the new solution found is better than the upcoming one, in which case it updates it. After that, the counter i, which counts the number of iterations without improvement, and the variable t, which counts the time since the execution began, are updated. When the stopping condition holds and therefore the main loop ends, line 17 the algorithm returns the best cover found.

5.2.2 Phase 1: building an initial solution

Algorithm 6 shows the pseudo-code for the iterative procedure `GraspPhase1`. The working cover W is initially empty and its objective function value w is set to 0. At each iteration, a new subset is randomly chosen from the *Restricted Candidate List* (RCL). At Line 1, we use the procedure `InitializeCandidates` to initialize the associative array $\overline{\mathcal{U}}$ that contains the subsets

to be selected to enter the current cover W. To be more clear, each subset $\overline{\mathcal{U}}_j$ is initialized with the subset U_j, $j \in N$ specified by *instance*, and after each update of the current partial cover W, $\overline{\mathcal{U}}_j$ will only include the elements of U_j that have not yet been covered by W. The RCL includes the p most promising subsets that were determined by ordering in non-decreasing value, the candidates based on the ratio's $\frac{R_j}{|\overline{\mathcal{U}}_j|}$. The order is adaptive because it depends on the size of $\overline{\mathcal{U}}_j$ (which only contains the elements of U_j not yet covered by the current partial cover W) and the value of R_j (includes the cost of the conflicts between the subset U_j and the subsets that already belong to the current partial cover W). In order to efficiently implement this strategy we need a specialized data structure (Z) that during each iteration keeps track of the repeated changes of the ratios that determines the RCL (using the procedure `BuildCandidateStructure` in line2).
Until the selected collection of subsets whose indexes are inserted into W provides a cover for U, the while loop (Lines 3–40) is repeated. The algorithm randomly chooses a subset from the RCL function `PopRandomCandidate`, adds it to the current cover W and removes it from $\overline{\mathcal{U}}$ (lines 4–6). The algorithm then determines if the current collection of subsets inside W has previously been met in previous iterations (Line 7). If so, the procedure obtains the information that was previously calculated from the *sharedCache* object and set the updated values of all data structures (Line 8), otherwise we update data structures accordingly (Lines 10–38). Until $\overline{\mathcal{U}}$ is empty, this operation is repeated, indicating that W is a cover of U. The procedure operates as follows each time the addition of a subset to the working cover W results in a never seen partial cover:

1. At line 10, The selected subset's cost R_j, is added to the working cover cost w;

2. The structure Z is updated by the current cost of those subsets that are in conflict with j, using the `UpdateCandidateStructure` function at lines 11–14, where the index of the subset that is in conflict with j is l;

3. The subsets that have become redundant after the addition of subset U_j are removed from W (Lines 15–29). To achieve this goal we note that a subset U_l inside the current cover is redundant when all its elements are covered by at least another subset (i.e., when set $\mathcal{Q}_l$ is empty). To this aim, we iterate on elements i in U_j. If i was previously uncovered, U_j is recorded as the unique subset that covers i (Lines 16–17); otherwise, if i was previously covered by only one other subset l in the working cover, i.e. $\mathcal{V}_i$ contains exactly one index (the value returned by `Peek`($\mathcal{V}_i$), we keep track that such subset is no more the only one that covers i (Lines 19–26). If this action makes U_l redundant, we add l to the list T of redundant subsets that may be eliminated (Lines 22–24). In any case, index j is added to the collection of the subsets in the working cover W that cover element i (Line 27). Finally, the `PruneWorkingCover` function actually removes from W some of the redundant subsets in T and updates all the other data structures accordingly (Line 29). Note that when T contains more than one redundant subset, they may have non-empty pairwise intersection. In turn, this may imply that only some of them can be removed without

uncovering some already covered elements. In this situation, making the best choice would require a cumbersome computational analysis of all the possibilities. However, since the goal of the current phase is to generate a feasible initial solution, we randomly remove as many redundant subsets as possible using a *first in first out* strategy and delegating any refinement to the second phase of the algorithm. The pseudo-code of function texttttPruneWorkingCover is provided at the end of Algorithm 6.

4. $\overline{\mathcal{U}}_j$ is subtracted from each subset $\overline{\mathcal{U}}_l$ inside $\overline{\mathcal{U}}$.If $\overline{\mathcal{U}}_j$ contains $\overline{\mathcal{U}}_l$ we remove l from $\overline{\mathcal{U}}$ and from Z, otherwise we replace $\overline{\mathcal{U}}_l$ with $\overline{\mathcal{U}}_l \setminus \overline{\mathcal{U}}_j$ (Lines 30–37).

5. The details regarding the new working cover w are stored in *sharedCache* object for potential future use.

The subsets in W offer a feasible solution with value w at the conclusion of `GraspPhase1`.

Algorithm 6: GraspPhase1

```
   Data: instance, C, I, W, w, R, V, Q, S, sharedCache
 1 Ū ← InitializeCandidates(instance) ;
 2 Z ← BuildCandidateStructure(S, Ū, R) ;
 3 while Ū ≠ ∅ do
 4 |   j ← PopRandomCandidate(Z) ;
 5 |   W ← W ∪ j;
 6 |   Ū ← Ū \ j ;
 7 |   if sharedCache_W ≠ null then
 8 |   |   (Ū, w, R, V, Q, Z) ← sharedCache_W ;
 9 |   else
10 |   |   w ← w + R_j ;
11 |   |   for l in C_j do
12 |   |   |   R_l ← R_l + d_jl;
13 |   |   |   Z ← UpdateCandidateStructure(Z, l, R_l/|Ū_l|) ;
14 |   |   end
15 |   |   T ← ∅  for i in U_j do
16 |   |   |   if |V_i| = 0 then
17 |   |   |   |   Q_j ← Q_j ∪ { i } ;
18 |   |   |   else
19 |   |   |   |   if |V_i| = 1 then
20 |   |   |   |   |   l ← Peek(V_i);
21 |   |   |   |   |   Q_l ← Q_l \ { i };
22 |   |   |   |   |   if |Q_l| = 0 then
23 |   |   |   |   |   |   T ← T ∪ { l };
24 |   |   |   |   |   end
25 |   |   |   |   end
26 |   |   |   end
27 |   |   |   V_i ← V_i ∪ { j } ;
28 |   |   end
29 |   |   PruneWorkingCover(instance, C, I, W, w, R, V, Q, Ū, Z, T);
30 |   |   for l in Ū do
31 |   |   |   if Ū_l \ Ū_j = ∅ then
32 |   |   |   |   Ū ← Ū \ l;
33 |   |   |   |   Z ← RemoveCandidate(Z, l) ;
34 |   |   |   else
35 |   |   |   |   Ū_l ← Ū_l \ Ū_j;
36 |   |   |   end
37 |   |   end
38 |   |   Put(sharedCache, W, (Ū, w, R, V, Q, Z)) ;
39 |   end
40 end
```

Algorithm 7: PruneWorkingCover

```
Data: instance, C, I, W, w, R, V, Q, Ū, Z, T
for j in T do
    if |Q_j| > 0 then
        W ← W \ { j };
        Q_j ← ∅;
        w ← w − R_j for i in U_j do
            V_i ← V_i \ { j };
            if |V_i| = 1 then
                l ← Peek(V_i);
                Q_l ← Q_l ∪ { i };
            end
        end
        for l in C_j do
            R_l ← R_l − d_jl;
            Z ← UpdateCandidateStructure(Z, l, R_l/|Ū_l|);
        end
    end
end
```

5.2.3 Phase 2: Local Search

Function `GraspPhase2` employs a sequential local search strategy (pseudo-code is shown in Algorithm 8).

Let $r, s \in \mathbb{N}$, we define a (r, s)-*exchange* move as a function that exchanges r subsets in the cover W with s subsets not belonging to W and preserves feasibility.
Let $M = \mathbb{N}^+ \times \{ 0, 1 \} \cup \{ 1 \}$, we define $Neigh_M(W)$ as the neighborhood of W containing all the covers that can be obtained from W applying once any (r, s)-exchange move with $(r, s) \in M$. Since M can be written as the disjoint union of the following subsets

$$
\begin{aligned}
M_0 &= \mathbb{N}^+ \times \{ 0 \}, \\
M_1 &= \mathbb{N}^+ \times \{ 1 \}, \\
M_2 &= \{ 1 \} \times \{ k \mid k \in \mathbb{N},\ k \geq 2 \}, \\
M &= M_0 \amalg M_1 \amalg M_2,
\end{aligned}
$$

we can partition $Neigh_M(W)$ into three smaller disjoint neighbourhoods $Neigh_0(W), Neigh_1(W)$ and $Neigh_2(W)$ corresponding to the (r, s)-exchange moves with (r, s) belonging to M_0, M_1 or M_2, respectively.

The proposed local search strategy consists of the following steps.

1. Explore $Neigh_0(W)$ using a best improvement strategy (function `ExploreNeigh0`). This neighborhood contains all the feasible covers of U that can be obtained removing one or more subsets from the current solution W. A necessary and sufficient condition to be able to remove a single subset U_j with index j from W is that the set of elements covered only by such subset must be empty, i.e. $\mathcal{Q}_j = \emptyset$. Let $F \subseteq W$ be the set of removable subsets belonging to the current cover that satisfy this condition. Note that it may not be possible to remove all the subsets in F at once. We apply a procedure to determine the subfamily of F that once removed maintains the feasibility and produces the highest decrease of the objective function value.

2. The function `ExploreNeigh1`, explore the neighborhood of $Neigh_1(W)$ and it is a $(r, 1)$-exchange. If we obtain a new cover that improves the incumbent solution, we update the best solution found and restart the search from step 1, then returning to explore the neighborhood of $Neigh_0(W)$.

3. Explore $Neigh_2(W)$ (function `ExploreNeigh2`). This neighborhood is defined by the set of $(1, s)$-exchange moves ($s \geq 2$), namely the moves that replace exactly one subset in the current cover W with two or more external subsets while preserving feasibility. Fully exploring all these possibilities may require a large amount of time since restoring feasibility after the removal of a subset from a feasible solution amounts to solve a smaller SCCS instance. As a consequence, we explore this neighbourhood only partially, starting from the most promising move and setting a time threshold τ to the evaluation of the moves that replace a given subset inside W (we forcefully stop the analysis when the time threshold is reached and move to the next one). More precisely, for each subset U_j, $j \in W$, we retrieve the set $\mathcal{Q}_j$ of elements covered only by U_j and, for each such an element, we sort its incident subsets by increasing current cost. We iterate over all possible Cartesian products of the subsets that cover the elements previously covered only by U_j and, if their addition to W improves the best value found so far, we save this move as the incumbent one. Once all subsets in W have been evaluated for removal, if we have found at least one cover improving the current one, we update W and restart from step 1, otherwise the local search ends.

Algorithm 8: GraspPhase2

```
Data: instance, C, I, W, w, R, V, Q
(W*, w*) ← (W, w);
exit ← false;
repeat
    (W, w) ← ExploreNeigh0(instance, C, I, W, w, R, V, Q);
    if w < w* then
        (W*, w*) ← (W, w);
    else
        (W, w) ← ExploreNeigh1(instance, C, I, W, w, R, V, Q);
        if w < w* then
            (W*, w*) ← (W, w);
        else
            (W, w) ← ExploreNeigh2(instance, C, I, W, w, R, V, Q);
            if w < w* then
                (W*, w*) ← (W, w);
            else
                exit ← true;
            end
        end
    end
until exit;
```

The procedure continues to iterate until the cover W can no longer be improved.

To efficiently implement the above procedure, we use information stored inside the associative arrays $\mathcal{V}$ and $\mathcal{Q}$. To remove a subset U_j, $j \in N$, from the current solution, a necessary and sufficient condition to restore the feasibility of the current cover W is to add a (possibly empty) collection of available subsets that covers the elements inside $\mathcal{Q}_j$. Data structures required during these steps are mainly needed to track the local changes of their global counterparts.

As far as time complexity is concerned, we observe that the size of the neighborhood in Step 1 depends on the cardinality of W which is at most $O(n)$ (although one may expect that on average $|W| \ll n$). To detect the best possible $(r, 0)$-exchange move, during each iteration the procedure goes through all the elements of a given subset and through all the subsets that have a non-zero conflict costs with such subset. Determine whether a subset can be added to the collection of removable subsets requires a time complexity of $O(s + t)$, where s is the maximum cardinality of a subset (bounded by m) and t is the maximum number of conflicts that involve a fixed subset in U (bounded by $n - 1$). $Neigh_1(W)$ contains all covers that can be obtained adding one external subset to W and removing the most convenient ones. Once the external subset is inserted, finding the best collection of subsets that can be removed preserving feasibility amounts to evaluate all possible $(r, 0)$-exchange moves and we apply the same strategy adopted for $Neigh_0(W)$. Finally, as already described, when exploring neighbourhood $Neigh_2(W)$ we

set a time limit for the evaluation of each collection of moves that removes a given subset from W preserving feasibility.

5.2.4 Parallel GRASP

Parallel processing can greatly improve the outcome of the single-process GRASP. Due to its randomly-restart nature, the higher the number of restarts and thus of feasible solutions the algorithm is able to evaluate, the higher the chance it has to improve the value of the final

solution.

Algorithm 9: GraspCoordinator

Data: $instance, i_{\max}, t_{\max}, cpuCount$
Result: Incumbent solution W^* and its objective value w^*
1 $(workers, inbound, outbound, sharedCache, incumbentCost) \leftarrow$ `InitMultiprocessing`$(cpuCount)$;
2 **forall** $worker$ ***in*** $workers$ **do**
3 `Run`$(worker,$ `GraspWorker`$,(instance, i_{\max}, t_{\max}, inbound, outbound, sharedCache, incumbentCost))$;
4 **end**
5 $S \leftarrow \emptyset$;
6 **forall** i ***in*** $1, \ldots, |workers|$ **do**
7 $W \leftarrow$ `Poll`$(outbound)$;
8 $S \leftarrow S \cup W$;
9 **end**
10 `Clear`$(sharedCache)$;
11 **forall** i ***in*** $1, \ldots, |workers|$ **do**
12 `Put`$(inbound, S)$;
13 **end**
14 $W^* \leftarrow N$;
15 $w^* \leftarrow +\infty$;
16 **forall** i ***in*** $1, \ldots, |workers|$ **do**
17 $(W, w) \leftarrow$ `Poll`$(outbound)$;
18 **if** $w < w^*$ **then**
19 $W^* \leftarrow W$;
20 $w^* \leftarrow w$;
21 **end**
22 **end**
23 **return** (W^*, w^*)

Algorithm 10: GraspWorker

Data: $instance, i_{\max}, t_{\max}, inbound, outbound, sharedCache, incumbentCost$
1 $(W, w) \leftarrow$ `Grasp`$(instance, i_{\max}, t_{\max}, N, sharedCache, incumbentCost)$;
2 `Put`$(outbound, W)$;
3 $S \leftarrow$ `Poll`$(inbound)$;
4 $(W, w) \leftarrow$ `Grasp`$(instance, i_{\max}, t_{\max}, S, sharedCache, incumbentCost)$;
5 `Put`$(outbound, (W, w))$;

A naive application of parallel computing would be to spawn as many processes as the number of available CPUs on the machine where the program is executed, each one carrying out

the algorithm previously described. After the termination of all executions, the main procedure would gather the individual result of each process and select the best solution among all the available ones. However, since each process executes the same optimization steps, it is beneficial to save and share among all processes the information computed during each execution (especially for computationally intensive tasks). This prevents each process from performing exactly the same operations that others have done before. This strategy can be usefully adopted during the first phase of the single-process GRASP. When building the initial feasible solution the procedure starts from an empty collection and iteratively add one subset randomly chosen from the RCL. Each time this action is carried out we are forced to update many data structures (namely $\overline{\mathcal{U}}$, W, w, R, $\mathcal{V}$, $\mathcal{Q}$ and Z) to reflect the effect of the inclusion of a new subset inside the cover that is currently being built. However, the outcome after each iteration of the first phase of the algorithm solely depends on which subsets belong to the partial cover that we have constructed so far. This is a type of information that can be easily shared among all processes. Therefore, in the parallel implementation of our algorithm, once the new state of each data structure has been computed for a given (partial) cover W, it is stored in a hash table ($sharedCache$) available to all the processes. In this way, when any iteration of the first phase of the single-process GRASP needs to evaluate an already analyzed partial cover W, no additional computational effort is required. Note that this may come out to be particularly convenient when we need to retrieve the state of the data structures associated with partial covers containing a few (typically of high quality) subsets: they have a high probability to appear more than once across different processes.

In order to reduce the overall running time, during the execution of the algorithm we also share among all workers the value of the best cover found at any given time. To this aim, we must replace Lines 9–14 in Algorithm 5 with the ones contained in Algorithm 11.

Algorithm 11: Parallel GRASP: change to Algorithm 5

```
 9 if w < incumbentCost then
10 |  (W*, w*, incumbentCost) ← (W, w, w) i ← 0
11 else
12 |  i ← i + 1
13 end
```

The benefits provided by multiprocessing are not limited to information sharing. The possibility to run the same instance multiple times in parallel has also been exploited to develop a *two stage algorithm*. Performing the same non-deterministic task in multiple processes leads to (most likely) different solutions. In our case a solution consists of a cover of U that, although not optimal, can be expected to contain at least some of the subsets that belong to an optimal solution. The union of all such solutions is likely to contain most (if not all) the subsets needed to build an optimal cover. Therefore, at the end of the first stage we collect the best solutions found by each process and during the first phase of the second stage we restrict the candidate list to the collection of subsets selected in this way. This is the goal of the data structure S that appears in the signature of function `Grasp` in Algorithm 5

An algorithm that leverages parallel processing needs a coordinator (at an outer layer) to handle and coordinate the work of each sub-process. The sole purpose of function `GraspCoordinator` in Algorithm 9, (from now on the *main process*) is to orchestrate the execution of a pool of sub-processes (from now on the *workers*) in charge of running function `Grasp` (Algorithm 5).

Function `GraspCoordinator` takes as input the instance data and the stopping rule parameters already discussed plus the integer value *cpuCount*, that represents the number of CPUs the algorithm can use during the execution. At Line 1, function `InitMultiprocessing` is called to instantiate the *workers*, two queue-like data structure (*inbound* and *outbound*) used to collect the individual worker result at the end of the first and second stage of the algorithm, the associative array *sharedCache*, accessible by all workers, used to share the output of the most expensive computations, and the best incumbent objective function value *incumbentCost* found among all workers up to a given instant of time.

At Lines 2–4, Algorithm 9 calls the `Run` function that instructs each worker (the first argument) to execute the single-process GRASP (the second argument) on the necessary input data (the third argument). At Lines 5–9, Algorithm 9 gathers the results of the first stage of each worker (the best covers found) and computes their union, storing the result inside the array S. Note that the main process (the `GraspCoordinator` function) will block waiting for all workers to put their results inside the *outbound* queue (Line 7). Once all workers have completed the first stage, the method exits the loop and goes to Lines 11–13 where the union of all the best covers found previously is added to the *inbound* queue (shared with each worker). Adding such information inside the *inbound* data structure provides the signal to each worker to start running the second stage of GRASP, using as available subsets for the first phase only those contained inside the set S. Eventually, at Lines 14–22, the algorithm waits for each worker to complete the execution of the second stage by querying the *outbound* queue. Once all results have been collected, the best one is returned as the final solution (Line 23).

5.2.5 Data structures and time and space complexity

Our single-process GRASP makes use of different data structures. Since most of the times we need to store and retrieve information associated to an element or a subset (both represented by an integer) the most common data structures (used many times in the algorithm) are specific kind of associative arrays, namely arrays and hash maps. Both these data structures are very well-known and we simply recall that they both provide $O(1)$ complexity when retrieving, inserting or updating an entry. However, in the first phase of our algorithm there is a step that requires to track the topmost p subsets ranked according to the value of $\frac{R_j}{|\overline{\mathcal{U}}_j|}$ (that depends on the current partial cover) as discussed in Section 5.2.2. This situation can not be effectively handled by using arrays or hash maps, since they do not provide efficient ways to sort their entries and to keep them sorted when the value of any of them changes. To address this issue we use a pair (H_1, H_2) of key-value binary heaps. The former is a maximum key-value binary heap with maximum size equal to p and at any given time during the execution of the algorithm it holds the subsets that have the lowest current costs. The latter, instead, is a minimum key-value binary heap that

contains all the remaining subsets (those with higher current costs). Whenever the current cost of any subset changes, we locate the heap that contains suck subset with $O(1)$ time complexity and update its value with $O(\log_2(p))$ time complexity if it belongs to H_1 and $O(\log_2(n-p))$ complexity if it belongs to H_2. Finally, we compare the top elements of H_1 and H_2 (which can be both retrieved in $O(1)$) and, if the former is greater than the latter, we swap them. At the end of this procedure H_1 will contain the p subsets with lowest current costs. One can easily check that in the worst-case scenario the time complexity of one iteration of the first phase of our algorithm is $O(n + m\log_2(n))$. Data structures involved in the second phase are just plain arrays and hash maps. As a final remark we note that adding the parallel processing to our algorithm has a space and time complexity that do not depend on the size of the input instance.

5.3 Experimental analysis

In this section, we first describe the data set and the algorithm parameters setting, then we provide the results obtained solving both mathematical formulations using Gurobi (with conflict threshold $k = 1$ and $k = 2$) and compare their performance with that of our parallel GRASP. The same machine has been used to execute all algorithms: an Intel Xeon Gold 6140M CPU 2.30GHz with 8 physical cores and 16 logical cores paired with 64 GB of RAM. The machine runs the operative system Microsoft Windows 10 Pro.

5.3.1 Parameters setting

The GRASP algorithm has been implemented using CPython 3.10. In particular, the parallel variant has been achieved using the `multiprocessing` module provided by the Python Standard Library. No other external software packages have been used. The main process in charge of handling the execution of all the subprocesses has been initialized with a number of workers equal to 15 (the number of available machine cores minus one used to handle the inter process communication workload and to provide to the operative system a free spot to execute all the other programs). We set GRASP parameters as follows:

- the size p of the RCL has been set to $\sqrt{n}$;
- the stopping rules parameters have been set to $i_{\max} = 50$ and $t_{\max} = 600$ seconds (equally divided between the first and second stage);
- the threshold time τ for the evaluation of each subfamily of moves in $Neigh_2(W)$ has been set to 0.01 seconds.

To solve the mathematical formulations, we used Gurobi 9.5 without changing its default parameters setting except for the *TimeLimit* parameter which has been set to 1 hour (in particular, the default value of the *MIPGap* parameter is 10^{-4}). As for GRASP, all logical cores have been made available to Gurobi that independently decided how many of them to use during the optimization process.

5.4 Instance generation

Since SCCS problem has not been previously studied in the literature, no benchmark instances exist. We decided to generate them by adapting the instances for the set covering problem made available by Beasley in the OR-library [Bea90a, Bea90b]. In Beasley's instances the size and the number of overlaps between subsets are not high enough to determine a consistent number of conflicts even when $k = 1$ (the value $\max\{\max_{j,l \in N, j \neq l}\{\,|U_j \cap U_l| - k\,\}, 0\}$ is often very low or equal to zero). For this reason, for each instance in the OR-library we generated a new instance obtained by merging three consecutive subsets (as appearing inside Beasley's original instance) into a single one with cost equal to the sum of the costs of the merged subsets. Thanks to this operation, the number of conflicts in each instance increased to a meaningful value for our purposes. As a consequence of this adaptation, the new instances contain a number of subsets that is about one third of the number of subsets in the original ones. Moreover, to compute the conflict costs we set

$$\gamma := \max\left\{ \left\lceil \max_{i \in U} \frac{c_i}{|U_i|} \right\rceil, 1 \right\}.$$

To distinguish the new generated instances from the original ones we append the suffix '*-3*' to the name of the former ones (e.g., the name of the original instance '*scp42*' from the OR-library becomes '*scp42-3*').
Computational tests have been carried out on 80 instances partitioned into three sets. *Set-A*, consisting of 50 instances, has been obtained by adapting the instances proposed in [Bea87]. *Set-B* contains 10 instances and has been obtained by modifying the instances proposed in [GW97]. Finally, *Set-C* has 20 instances obtained by merging instances proposed in [Bea92]. The main features of these three sets are summarized in Tables 5.1-5.2 where the first column shows the name of the instance, the second and third columns show the number of elements ($|U|$) and of subsets ($|N|$), whereas the last two columns show the number of pairs of subsets in conflict when $k = 1$ and $k = 2$, respectively.

5.4.1 Computational results

In Table 5.3 we compare the performance of the two mathematical formulations (SCCS_BIN and SCCS_MIP) solved with Gurobi. Since we set a time limit, Gurobi's execution may terminate with either an `OPTIMAL` status (when an optimal solution has been found) or a `TIME LIMIT` status (when the 1 hour time limit has been reached and the incumbent feasible solution is returned). The table is partitioned into two parts, one devoted to the comparison of the two models when $k = 1$ and the other when $k = 2$. For each model, column *Obj* provides the solution value, column *ObjC* indicates the sum of all conflict costs and shows into brackets the corresponding number of conflicts *#conf.* Column *Time* contains the computational time in seconds. Finally, the last two lines of the table report the average objective and time values *Avg* and the number of optimal solutions *#Opt* found by the two models out of all instances, respectively. When analyzing these last two lines, it is evident how both mathematical formulations allow to obtain

the same number of optimal solutions when $k = 1$ and $k = 2$ and that such number is larger for $k = 2$ (29 instances solved to optimality vs. 18). Moreover, the average computational time decreases from around 2940 seconds, for $k = 1$, to around 2340 seconds, for $k = 2$. Thus, instances with $k = 2$ seem to be easier to solve than instances with $k = 1$: this is probably due to the lower number of conflicts in the instances when $k = 2$ (cf. Tables 5.1-5.2). Instances in *Set-C* are the ones with the highest number of conflicts and this is, probably, the reason why no optimal solutions have been found by Gurobi in any of these instances.

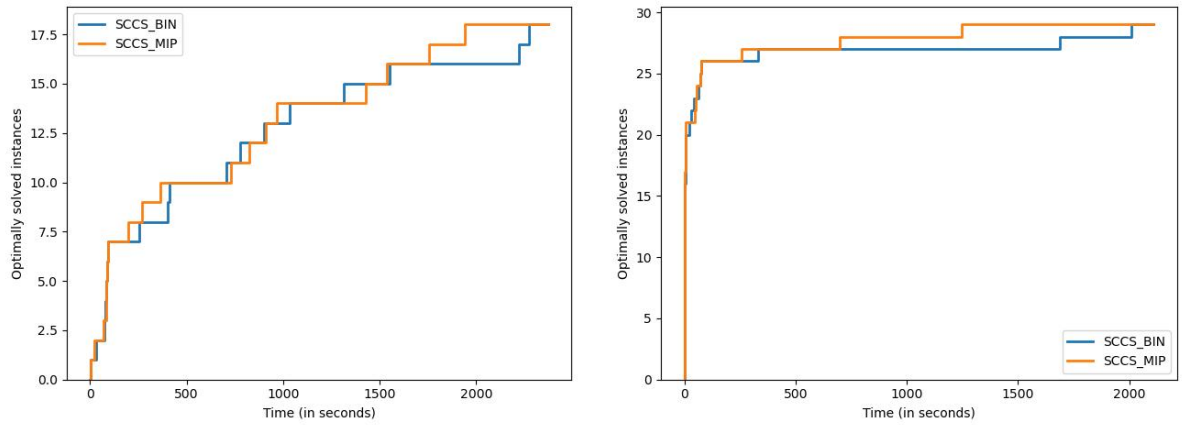


Figure 5.1: SCCS_BIN vs. SCCS_MIP: percentage of instances solved to optimality for (a) $k = 1$ and (b) $k = 2$.

In Figure 5.1, we show the results reported in Table 5.3 in a way that better highlights the performance of the models. In Figure 5.1, we show the results reported in Table 5.3 in a way that better highlights the performance of the models. The horizontal axis reports the computational time in seconds and the vertical one the number of instances optimally solved within that time. In other words, the (x, y) point on this plot shows the percentage of optimally solved instances (y value) in less than or equal to x seconds. This implies that the faster the growth of a curve, the better the performance. The blue curve is associated with SCCS_BIN model, whereas the orange one with SCCS_MIP. Figure 5.1a certifies the similar performance of the two models that we have already observed. However, we note that, overall, around 23% of the instances are solved to optimality by SCCS_BIN within 2280 seconds, while SCCS_MIP reaches the same result in 1950 seconds. This means that SCCS_MIP is 15% faster than SCCS_BIN in reaching the optimal solution. This trend is even more clear in Figure 5.1b. Here, the number of optimal solutions found is equal to 36% and most of them are found by both models in less than 75 seconds. However, the performance gap between the two models for $k = 2$ is more evident as SCCS_MIP reaches the highest percentage in about 1250 seconds while SCCS_BIN requires around 2000 seconds to achieve the same result. To summarize, the effectiveness of the two models is the same, but SCCS_MIP is, on average, more efficient than SCCS_BIN.

In Table 5.4 we compare the solutions found by our parallel variant of GRASP with the ones found by the two models. Also in this case the table is partitioned into two parts providing results for $k = 1$ and $k = 2$, respectively. Column *Gurobi* shows the best (possibly optimal) solution found by Gurobi within the time limit of 1 hour. Optimal solutions are emphasized by a trailing asterisk. Columns *ObjB* and *ObjMed* report the best and the median objective values of parallel GRASP out of 10 runs for each instance. The next four columns provide statistics on the best solution out of the ten runs: column *ObjC* shows the sum of all conflict costs and the corresponding number of conflicts *#conf*; column *TBest* reports the time to best that represents the time in seconds required to find the best solution, column *Time* shows the total computational time in seconds and, finally, the *Gap* column provides the percentage gap between *ObjB* and *Gurobi* values. This percentage value is computed as $100 \times \frac{Obj - Gurobi}{Gurobi}$. In each row, the best value is marked in bold. At the bottom of the table, the *Avg* row reports the average values of the computational time and of the percentage gap, while *#Best* shows how many times parallel GRASP finds a solution that is better than or equal to the one found by Gurobi. Finally, the row *AvgRSD* contains the average value of the relative standard deviation of the objective value computed using the results of the 10 runs. The results in row *#Best* show that parallel GRASP is extremely effective finding, for $k = 1$, in 75 out of 80 instances a solution better than or equal to the best one (the value is strictly better in 55 instances with an average improvement of 40.8%); on the remaining 5 instances, the percentage gap is lower than 3.7%. Similar results are observed for $k = 2$ where for 76 instances the solution provided by parallel GRASP is the best one (the value is strictly better in 44 instances with an average improvement of 46.0%); the percentage gap in the remaining 4 instances is lower than 3.8%. It is worth noting that GRASP always finds the optimal solution in the 47 instances where this solution is known with the only exception of instances *scp41-3* and *scp410-3* for $k = 1$. Moreover, the average percentage gaps equal to -28% for $k = 1$ and -23% for $k = 2$ further highlight the effectiveness of our algorithm. Finally, the standard deviation values show that our algorithm is also stable with an average relative standard deviation equal to 1.65% and 1.22%, respectively.

The results of the *Gap* column confirm that the hardest instances to solve using the mathematical formulations are the ones in *Set-C*. Indeed, in these instances parallel GRASP finds solutions that are from 38% to 58% better than the solutions provided by Gurobi for $k = 1$, and from 41% to 69% for $k = 2$.
Finally, regarding the computational time, GRASP requires on average less than 400 seconds for $k = 1$ and less than 380 seconds for $k = 2$. In more than half of the instances the algorithm stops before reaching the time limit of 600 seconds.

We conclude this section providing some charts that allow to better understand the contribution of the different components of the GRASP algorithm and to highlight how the structural differences of the test instances at hand reflect on the internal workload.

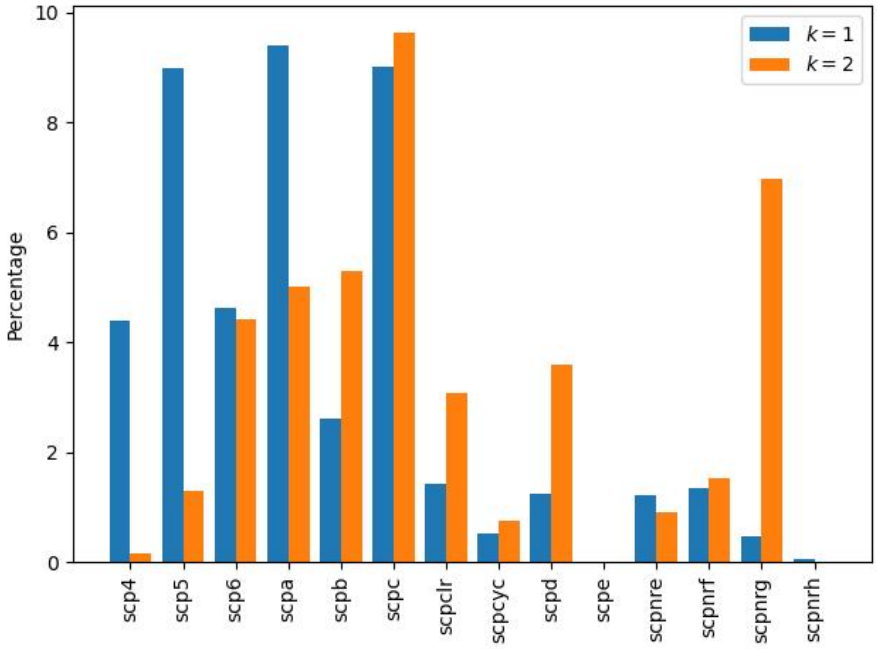


Figure 5.2: Average percentage improvement of incumbent solution value between one stage and two stage GRASP.

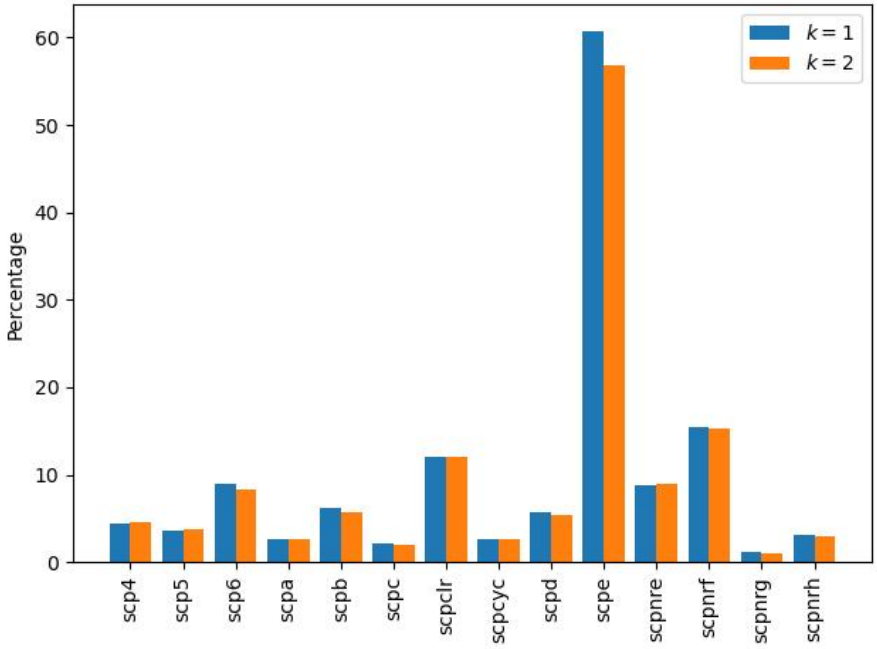
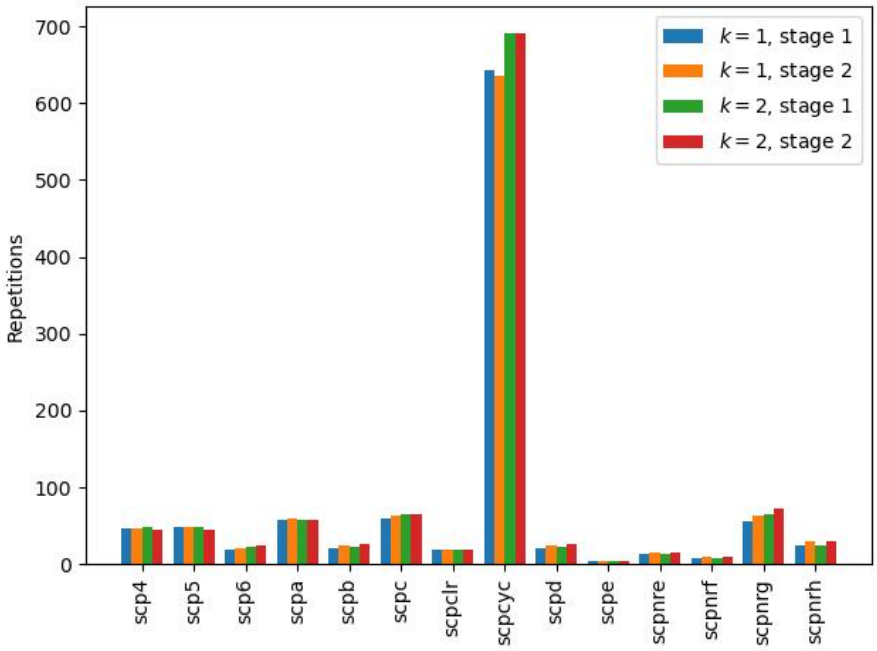
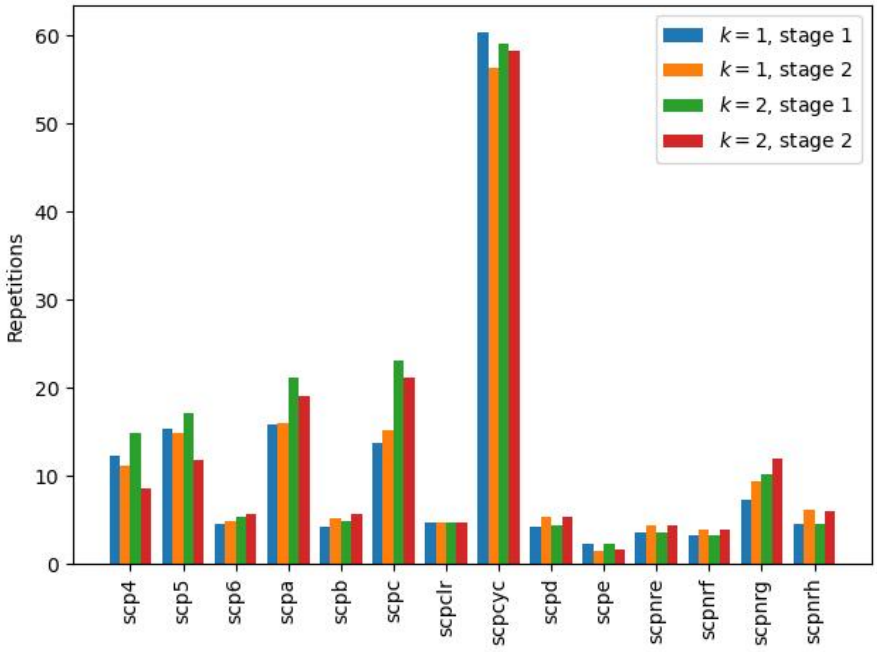


Figure 5.3: Average percentage number of times precomputed data has been retrieved from the shared cache.



(a) Average number of repetitions of phase 1



(b) Average number of repetitions of phase 2

Figure 5.4: Average number of repetitions of each phase of the GRASP algorithm

Figure 5.2 shows the performance impact of the second stage (when GRASP is rerun restricting the subsets taken into account during the first phase) that consistently improves the value of the final solution. Each bar represents the percentage improvement of the average solution value computed on the instances of a given family (each instance has been run ten times). The family that benefits the most by the second stage is the *scpc* family that has an average improvement of 9%. A notable exception is given by the *scpe* family that never improves the final solution during the second stage: this is due to the fact that the instances of this family are the smallest ones (with $|U| = 50$ and $|\mathcal{U}| = 167$) and both Gurobi and our heuristic algorithm always find the optimal solutions within a short computing time.

Figure 5.3 quantifies the benefit yielded from the adoption of the shared cache objects during the first phase (see Section 5.2.2). Each bar of the Figure represents for each family of instances the average percentage number of times when parallel GRASP has been able to retrieve data from the shared cache object avoiding further computations (each instance has been run ten times). Again, the particular nature of *scpe* family leads to an extremely high number of times that the GRASP algorithm reuses previously computed information. For all the other families this value ranges between 1% and 15% (recall that each time the required information is retrieved from the shared cache object the algorithm is able to replace a procedure with temporal cost $O(n + m \log_2 n)$ with a simple read from memory with cost $O(1)$).

Figures 5.4a and 5.4b show how many times the main loop of the first and second phase of the GRASP algorithm have been repeated on average in order to complete a single iteration of the GRASP algorithm. Recall that one iteration of the main loop of the first phase of the parallel GRASP amounts to the addition of a subset to the current partial cover, while one iteration of the main loop of the second phase of the algorithm corresponds to searching the neighbourhood of the current solution for a better cover. Each bar of the figure shows the average number of times that a given phase has been repeated across all instances belonging to a given family (each instance has been run ten times). The particularly high values of the *scpcyc* family in Figures 5.4a and 5.4b are due to the fact that each subset U_j that belongs to the instances of this family has at most 5 elements and the total number of elements of U ranges from 240 in instance *scpcyc06-3* up to 28160 in instance *scpcyc11-3*.

Instance	$\|U\|$	$\|\mathcal{U}\|$	#conf ($k=1$)	#conf ($k=2$)
Set-A				
scp41-3	200	334	8351	1908
scp42-3	200	334	8306	1848
scp43-3	200	334	8258	1791
scp44-3	200	334	8596	1977
scp45-3	200	334	8136	1789
scp46-3	200	334	8826	2055
scp47-3	200	334	7843	1711
scp48-3	200	334	8526	1860
scp49-3	200	334	8195	1807
scp51-3	200	667	33962	8238
scp52-3	200	667	34092	8047
scp53-3	200	667	34153	8104
scp54-3	200	667	33893	7913
scp55-3	200	667	32490	7376
scp56-3	200	667	34551	8355
scp57-3	200	667	34832	8393
scp58-3	200	667	33285	7784
scp59-3	200	667	33094	7597
scp61-3	200	334	49416	40825
scp62-3	200	334	49916	41855
scp63-3	200	334	49827	41583
scp64-3	200	334	49404	40723
scp65-3	200	334	49830	41733
scp410-3	200	334	7944	1793
scp510-3	200	667	34476	8321
scpa1-3	300	1000	139239	49629
scpa2-3	300	1000	139174	49455
scpa3-3	300	1000	139351	49665
scpa4-3	300	1000	139104	50086
scpa5-3	300	1000	139255	49635
scpb1-3	300	1000	485873	457826
scpb2-3	300	1000	486022	458275
scpb3-3	300	1000	485654	457218
scpb4-3	300	1000	485276	456577
scpb5-3	300	1000	485595	457456
scpc1-3	400	1334	347270	152214
scpc2-3	400	1334	345511	150739
scpc3-3	400	1334	345621	151469
scpc4-3	400	1334	346553	150846
scpc5-3	400	1334	346137	151744
scpd1-3	400	1334	883245	867920
scpd2-3	400	1334	883605	869019
scpd3-3	400	1334	883254	868280
scpd4-3	400	1334	883136	868196
scpd5-3	400	1334	883471	868775
scpe1-3	50	167	13810	13707
scpe2-3	50	167	13837	13756
scpe3-3	50	167	13755	13666
scpe4-3	50	167	13818	13707
scpe5-3	50	167	13839	13790

Table 5.1: Benchmark instances: Set A

Instance	$\|U\|$	$\|\mathcal{U}\|$	#conf ($k=1$)	#conf ($k=2$)
Set-B				
scpclr10-3	511	70	2415	2415
scpclr11-3	1023	110	5995	5995
scpclr12-3	2047	165	13530	13530
scpclr13-3	4095	239	28441	28441
scpcyc06-3	240	64	281	153
scpcyc07-3	672	150	806	443
scpcyc08-3	1792	342	2173	1230
scpcyc09-3	4608	768	5660	3267
scpcyc10-3	11520	1707	14370	8401
scpcyc11-3	28160	3755	35487	21072
Set-C				
scpnre1-3	500	1667	1388611	1388611
scpnre2-3	500	1667	1388611	1388611
scpnre3-3	500	1667	1388611	1388611
scpnre4-3	500	1667	1388611	1388611
scpnre5-3	500	1667	1388611	1388611
scpnrf1-3	500	1667	1388611	1388611
scpnrf2-3	500	1667	1388611	1388611
scpnrf3-3	500	1667	1388611	1388611
scpnrf4-3	500	1667	1388611	1388611
scpnrf5-3	500	1667	1388611	1388611
scpnrg1-3	1000	3334	4628203	3574676
scpnrg2-3	1000	3334	4626736	3574156
scpnrg3-3	1000	3334	4623974	3574347
scpnrg4-3	1000	3334	4630196	3579650
scpnrg5-3	1000	3334	4625519	3575142
scpnrh1-3	1000	3334	5555947	5555602
scpnrh2-3	1000	3334	5555936	5555589
scpnrh3-3	1000	3334	5555946	5555566
scpnrh4-3	1000	3334	5555937	5555574
scpnrh5-3	1000	3334	5555925	5555576

Table 5.2: Benchmark instances: Set B and Set C

	GUROBI											
	k=1						k=2					
	SCCS_BIN			SCCS_MIP			SCCS_BIN			SCCS_MIP		
Instance	Obj	ObjC (#conf)	Time	Obj	ObjC (#conf)	Time	Obj	ObjC (#conf)	Time	Obj	ObjC (#conf)	Time
Set-A												
scp41-3	2037*	350 (7)	253.69	2037*	350 (7)	200.33	1108*	50 (1)	1.50	1108*	50 (1)	1.47
scp42-3	1977*	700 (14)	81.81	1977*	700 (14)	84.44	1209*	100 (2)	22.47	1209*	100 (2)	1.42
scp43-3	2583*	500 (5)	1314.28	2583*	500 (5)	1428.64	1113*	0 (0)	0.86	1113*	0 (0)	0.94
scp44-3	2543*	686 (7)	1033.85	2543*	686 (7)	824.51	1192*	98 (1)	2.64	1192*	98 (1)	2.62
scp45-3	2247*	413 (7)	400.11	2247*	413 (7)	363.55	1279*	59 (1)	2.00	1279*	59 (1)	1.78
scp46-3	2602*	666 (9)	1550.40	2602*	666 (9)	1535.76	1302*	74 (1)	3.23	1302*	74 (1)	3.22
scp47-3	2128*	590 (8)	778.80	2128*	590 (8)	731.50	1116*	118 (2)	2.36	1116*	118 (2)	2.30
scp48-3	2647*	567 (7)	2220.35	2647*	567 (7)	1756.22	1149*	0 (0)	1.69	1149*	0 (0)	1.53
scp49-3	2604*	564 (6)	705.27	2604*	564 (6)	968.81	1398*	0 (0)	4.83	1398*	0 (0)	4.83
scp51-3	1532	304 (4)	3601.05	1615	76 (1)	3600.84	618*	0 (0)	4.94	618*	0 (0)	5.39
scp52-3	1547	219 (3)	3600.77	1446	73 (1)	3606.41	602*	0 (0)	1.19	602*	0 (0)	1.20
scp53-3	1506	300 (3)	3600.72	1423	0 (0)	3602.14	627*	0 (0)	5.14	627*	0 (0)	5.36
scp54-3	1501	196 (2)	3601.05	1360	196 (2)	3601.83	546*	0 (0)	2.14	546*	0 (0)	2.09
scp55-3	1375	201 (3)	3601.73	1375	201 (3)	3602.23	528*	0 (0)	0.78	528*	0 (0)	0.81
scp56-3	1410	91 (1)	3601.16	1410	91 (1)	3601.00	511*	0 (0)	0.56	511*	0 (0)	0.47
scp57-3	1551	0 (0)	3601.06	1551	0 (0)	3601.41	764*	0 (0)	7.58	764*	0 (0)	7.73
scp58-3	1574	77 (1)	3600.43	1662	231 (3)	3600.60	650*	0 (0)	3.45	650*	0 (0)	3.34
scp59-3	1518	200 (2)	3601.19	1535	100 (1)	3601.18	660*	0 (0)	3.23	660*	0 (0)	3.34
scp61-3	5608	3572 (84)	3601.09	4846	3078 (74)	3601.16	2537	1406 (42)	3600.55	2462	1653 (50)	3600.55
scp62-3	4998	3211 (67)	3601.06	4379	2850 (63)	3601.03	2805	1520 (44)	3600.60	2805	1520 (44)	3600.42
scp63-3	6455	3900 (77)	3601.16	6227	4150 (69)	3601.12	3103	2000 (41)	3600.47	3103	2000 (41)	3600.66
scp64-3	4905	3880 (84)	3600.97	4905	3880 (84)	3601.08	2554	1760 (47)	3600.59	2905	1980 (52)	3600.78
scp65-3	4508	3145 (73)	3601.05	4636	3230 (78)	3601.02	2512	1785 (51)	3600.54	2590	1496 (46)	3600.67
scp410-3	2501*	290 (5)	903.23	2501*	290 (5)	911.93	1404*	0 (0)	2.81	1404*	0 (0)	2.31
scp510-3	1547	178 (2)	3600.70	1547	178 (2)	3601.01	642*	0 (0)	1.86	642*	0 (0)	1.87
scpa1-3	4784	2080 (34)	3600.95	4784	2080 (34)	3601.44	963	52 (1)	3602.77	963	52 (1)	3603.54
scpa2-3	4516	2006 (29)	3601.14	4395	1947 (26)	3601.40	1048	59 (1)	3601.59	1071	118 (2)	3602.25
scpa3-3	4189	1500 (26)	3601.31	4189	1500 (26)	3600.96	910	0 (0)	3604.48	910	0 (0)	3604.49
scpa4-3	4242	952 (14)	3601.00	4242	952 (14)	3601.74	946	0 (0)	3602.87	946	0 (0)	3603.09
scpa5-3	5423	2077 (27)	3600.82	5000	2278 (31)	3600.82	894*	0 (0)	2010.28	894*	0 (0)	697.61
scpb1-3	10857	8853 (115)	3600.14	10857	8853 (115)	3600.13	6697	5772 (140)	3600.17	6697	5772 (140)	3600.14
scpb2-3	10581	9408 (235)	3600.12	10581	9408 (235)	3600.11	6109	5432 (178)	3600.17	6109	5432 (178)	3600.15
scpb3-3	9324	8772 (160)	3600.13	9324	8772 (160)	3600.13	5864	5112 (182)	3600.82	5864	5112 (182)	3600.88
scpb4-3	9576	8820 (257)	3600.11	9576	8820 (257)	3600.13	5145	4596 (151)	3600.42	5145	4596 (151)	3600.46
scpb5-3	10922	9758 (217)	3600.12	10922	9758 (217)	3600.13	6178	5614 (174)	3600.11	6178	5614 (174)	3600.14
scpc1-3	17478	13100 (96)	3601.28	17478	13100 (96)	3600.94	1771	0 (0)	3633.25	1645	0 (0)	3602.65
scpc2-3	7473	4785 (111)	3601.70	7473	4785 (111)	3602.05	1744	396 (12)	3629.31	1744	396 (12)	3638.22
scpc3-3	10699	7100 (107)	3601.10	10699	7100 (107)	3602.40	1625	100 (2)	3601.88	1625	100 (2)	3602.29
scpc4-3	7997	5206 (107)	3601.81	7997	5206 (107)	3601.83	1636	190 (5)	3626.54	1636	190 (5)	3629.82
scpc5-3	10557	7150 (105)	3601.31	10557	7150 (105)	3601.92	1640	100 (2)	3627.07	1640	100 (2)	3603.43
scpd1-3	12601	10845 (136)	3600.19	12601	10845 (136)	3600.22	8882	8181 (241)	3600.21	8882	8181 (241)	3600.23
scpd2-3	11682	10746 (304)	3600.25	11682	10746 (304)	3600.20	9258	8361 (279)	3600.21	9258	8361 (279)	3600.23
scpd3-3	11049	9432 (152)	3600.22	11049	9432 (152)	3600.21	9181	7218 (136)	3600.17	9181	7218 (136)	3600.22
scpd4-3	13535	12320 (120)	3600.39	13535	12320 (120)	3600.22	13738	12586 (231)	3600.18	13738	12586 (231)	3600.21
scpd5-3	13176	12366 (329)	3600.22	13176	12366 (329)	3600.24	8403	7659 (237)	3600.18	8403	7659 (237)	3600.20
scpe1-3	26*	17 (3)	72.41	26*	17 (3)	71.23	23*	14 (3)	69.64	23*	14 (3)	50.59
scpe2-3	28*	19 (3)	94.54	28*	19 (3)	93.20	25*	16 (3)	63.08	25*	16 (3)	71.44
scpe3-3	24*	18 (1)	33.94	24*	18 (1)	23.81	23*	11 (5)	42.39	23*	11 (5)	54.05
scpe4-3	25*	16 (3)	88.23	25*	16 (3)	88.88	22*	13 (3)	30.17	22*	13 (3)	45.31
scpe5-3	28*	19 (3)	85.77	28*	19 (3)	85.64	25*	16 (3)	73.84	25*	16 (3)	73.78
Set-B												
scpclr10-3	1926*	1893 (55)	412.10	1926*	1893 (55)	271.49	1871*	1838 (55)	331.91	1871*	1838 (55)	258.75
scpclr11-3	4215	4182 (55)	3600.43	4394	4358 (66)	3600.62	4034	3998 (66)	3600.26	3890	3854 (66)	3600.25
scpclr12-3	10143	10104 (78)	3601.17	11148	11106 (91)	3600.27	7728	7692 (66)	3600.23	10573	10534 (78)	3600.68
scpclr13-3	29504	29462 (91)	3600.45	25138	25098 (91)	3600.24	31803	31766 (78)	3600.17	29272	29224 (120)	3600.16
scpcyc06-3	126*	42 (27)	1.23	126*	42 (27)	2.41	99*	15 (9)	0.83	99*	15 (9)	0.72
scpcyc07-3	335*	136 (74)	2273.54	335*	136 (74)	1942.80	250*	49 (42)	1688.08	250*	49 (38)	1246.73
scpcyc08-3	911	434 (183)	3600.78	919	442 (176)	3600.22	662	182 (160)	3600.16	668	185 (152)	3600.16
scpcyc09-3	2332	1234 (485)	3600.83	2748	1635 (948)	3601.43	1678	565 (453)	3600.27	1721	608 (476)	3600.16
scpcyc10-3	6755	4214 (1966)	3600.70	7770	5145 (2742)	3600.74	4462	1966 (1136)	3600.58	4441	1909 (1376)	3600.38
scpcyc11-3	19422	13539 (6650)	3600.28	24296	17958 (9511)	3600.27	12771	6888 (4124)	3600.18	14998	8591 (5837)	3600.17
Set-C												
scpnre1-3	7424	7149 (78)	3600.35	7424	7149 (78)	3600.35	7047	5724 (55)	3600.29	7047	5724 (55)	3600.38
scpnre2-3	9054	8826 (78)	3618.44	9054	8826 (78)	3600.36	8820	8592 (78)	3600.31	8820	8592 (78)	3600.35
scpnre3-3	8211	7440 (105)	3600.28	8241	7551 (105)	3600.36	6581	6213 (78)	3601.35	6581	6213 (78)	3600.79
scpnre4-3	7230	7158 (78)	3601.18	7230	7158 (78)	3601.00	6327	6273 (66)	3600.33	6327	6273 (66)	3600.37
scpnre5-3	7488	6288 (55)	3600.30	7488	6288 (55)	3600.41	7323	6123 (55)	3600.32	7323	6123 (55)	3600.37
scpnrf1-3	2145	1608 (15)	3601.12	2145	1608 (15)	3601.24	2798	2777 (21)	3601.01	2798	2777 (21)	3601.10
scpnrf2-3	2324	1748 (15)	3601.03	2366	1736 (15)	3601.14	2276	1835 (15)	3601.17	2276	1835 (15)	3601.05
scpnrf3-3	2241	1836 (15)	3601.09	2241	1836 (15)	3601.08	2228	1865 (15)	3601.11	2503	1777 (15)	3610.04
scpnrf4-3	2484	2370 (21)	3601.12	2609	2524 (21)	3601.99	2657	1885 (15)	3601.02	2440	1725 (15)	3601.07
scpnrf5-3	2591	2425 (21)	3616.89	2499	1665 (15)	3601.20	2327	2048 (21)	3601.00	2328	1944 (21)	3601.11
scpnrg1-3	33276	32210 (1125)	3600.98	33276	32210 (1125)	3601.00	21735	21240 (794)	3600.78	21735	21240 (794)	3601.28
scpnrg2-3	34828	34560 (1039)	3601.02	34828	34560 (1039)	3601.03	15524	14020 (770)	3600.75	15524	14020 (770)	3600.75
scpnrg3-3	28168	26620 (1104)	3601.02	28168	26620 (1104)	3601.07	17639	15906 (771)	3600.69	17639	15906 (771)	3600.82
scpnrg4-3	30665	28740 (1324)	3600.96	30665	28740 (1324)	3601.42	17604	15940 (819)	3600.72	17604	15940 (819)	3601.21
scpnrg5-3	26931	24960 (1234)	3600.95	26931	24960 (1234)	3601.07	15825	15270 (755)	3602.36	15825	15270 (755)	3601.19
scpnrh1-3	19313	15684 (253)	3601.29	19313	15684 (253)	3601.54	20124	20019 (351)	3602.75	20124	20019 (351)	3601.32
scpnrh2-3	20580	20436 (351)	3601.26	18537	14442 (253)	3601.48	19527	19383 (351)	3601.27	19704	19566 (351)	3601.35
scpnrh3-3	17982	17889 (300)	3601.29	17982	17889 (300)	3601.40	18873	15234 (253)	3601.30	18873	15234 (253)	3601.35
scpnrh4-3	16749	14646 (253)	3601.27	16749	14646 (253)	3601.37	16815	16716 (300)	3601.39	16815	16716 (300)	3601.69
scpnrh5-3	17361	17268 (300)	3601.26	17361	17268 (300)	3601.43	16461	16368 (300)	3601.31	16461	16368 (300)	3601.34
Avg	7861.38		2944.89	7849.26		2933.15	5231.85		2351.80	5268.25		2328.50
#Opt			18			18			29			29

Table 5.3: Computational results of SCCS_BIN and SCCS_MIP.

	GRASP													
	k=1							k=2						
Instance	Best/Opt	ObjB	ObjMed	objC (#conf)	TBest	Time	Gap	Best/Opt	ObjB	ObjMed	objC (#conf)	TBest	Time	Gap
Set-A														
scp41-3	**2037***	2097	2097.00	650 (13)	79.00	119.25	2.95%	1108*	**1108**	1108.00	50 (1)	3.09	73.83	0.00%
scp42-3	1977*	**1977**	1977.00	700 (14)	15.95	106.19	0.00%	1209*	**1209**	1209.00	50 (1)	5.08	98.97	0.00%
scp43-3	2583*	**2583**	2613.00	500 (5)	60.25	113.95	0.00%	1113*	**1113**	1113.00	0 (0)	8.22	79.70	0.00%
scp44-3	2543*	**2543**	2575.00	686 (7)	3.88	111.17	0.00%	1192*	**1192**	1192.00	98 (1)	2.25	93.08	0.00%
scp45-3	2247*	**2247**	2247.00	413 (7)	4.70	97.88	0.00%	1279*	**1279**	1279.00	59 (1)	2.36	85.21	0.00%
scp46-3	2602*	**2602**	2650.50	666 (9)	74.17	115.00	0.00%	1302*	**1302**	1302.00	74 (1)	22.94	111.15	0.00%
scp47-3	2128*	**2128**	2128.00	590 (8)	4.84	107.97	0.00%	1116*	**1116**	1116.00	118 (2)	4.23	78.84	0.00%
scp48-3	2647*	**2647**	2673.00	567 (7)	33.81	130.66	0.00%	1149*	**1149**	1149.00	0 (0)	2.50	93.55	0.00%
scp49-3	2604*	**2604**	2658.00	564 (6)	90.04	128.35	0.00%	1398*	**1398**	1398.00	0 (0)	62.16	103.25	0.00%
scp51-3	1532	**1466**	1645.50	152 (2)	54.69	256.65	-4.31%	618*	**618**	618.00	0 (0)	3.34	154.99	0.00%
scp52-3	1446	**1446**	1496.00	73 (1)	164.63	249.99	0.00%	602*	**602**	602.00	0 (0)	1.81	133.85	0.00%
scp53-3	1423	**1423**	1497.00	0 (0)	157.61	257.41	0.00%	627*	**627**	627.00	0 (0)	34.41	180.89	0.00%
scp54-3	1360	**1360**	1471.00	196 (2)	149.83	247.57	0.00%	546*	**546**	546.00	0 (0)	86.25	140.69	0.00%
scp55-3	**1375**	1382	1429.00	335 (5)	176.09	268.44	0.51%	528*	**528**	528.00	0 (0)	7.34	127.00	0.00%
scp56-3	**1410**	1457	1534.00	182 (2)	177.64	266.15	3.33%	511*	**511**	511.00	0 (0)	4.98	105.06	0.00%
scp57-3	1551	**1551**	1646.50	0 (0)	242.93	339.66	0.00%	764*	**764**	764.00	0 (0)	16.17	159.22	0.00%
scp58-3	1574	**1517**	1610.00	231 (3)	160.74	257.72	-3.62%	650*	**650**	650.00	0 (0)	7.03	157.00	0.00%
scp59-3	**1518**	1574	1597.50	100 (1)	221.89	309.58	3.69%	660*	**660**	660.00	0 (0)	10.88	153.91	0.00%
scp61-3	4846	**2930**	3029.50	2166 (63)	7.59	63.25	-39.54%	2462	**1733**	1733.00	969 (30)	35.61	67.94	-29.61%
scp62-3	4379	**3327**	3468.00	2337 (58)	16.75	63.70	-24.02%	2805	**2037**	2037.00	1482 (52)	31.39	62.17	-27.38%
scp63-3	6227	**4105**	4406.00	3125 (70)	39.03	67.14	-34.08%	3103	**2485**	2573.00	1325 (34)	58.02	86.91	-19.92%
scp64-3	4905	**3211**	3382.50	2460 (63)	48.58	75.66	-34.54%	2554	**1899**	1899.00	940 (32)	2.39	68.44	-25.65%
scp65-3	4508	**3138**	3223.00	2142 (65)	1.02	55.67	-30.39%	2512	**1979**	1994.50	1020 (38)	3.25	68.64	-21.22%
scp410-3	**2501***	2511	2515.00	522 (9)	69.59	111.25	0.40%	1404*	**1404**	1404.00	0 (0)	5.66	98.97	0.00%
scp510-3	1547	**1338**	1435.00	178 (2)	233.04	321.06	-13.51%	642*	**642**	642.00	0 (0)	97.39	154.60	0.00%
scpa1-3	4784	**2901**	3056.50	1144 (19)	325.19	511.34	-39.36%	**963**	976	1006.00	52 (1)	327.52	469.33	1.35%
scpa2-3	4395	**2842**	3247.00	1062 (17)	413.80	-	-35.34%	1048	**1048**	1101.50	59 (1)	95.30	431.34	0.00%
scpa3-3	4189	**2691**	2842.00	1050 (20)	268.83	448.55	-35.76%	910	**910**	933.00	0 (0)	331.54	494.48	0.00%
scpa4-3	4242	**2888**	2990.00	680 (9)	403.96	-	-31.92%	946	**946**	954.00	0 (0)	225.69	399.25	0.00%
scpa5-3	5000	**2980**	3192.00	804 (11)	346.36	553.25	-40.40%	894*	**894**	894.00	0 (0)	257.72	421.49	0.00%
scpb1-3	10857	**4586**	5018.50	3692 (98)	255.92	373.42	-57.76%	6697	**3159**	3254.00	2535 (94)	244.88	389.52	-52.83%
scpb2-3	10581	**4925**	5214.50	3962 (98)	70.36	289.29	-53.45%	6109	**3336**	3470.00	2408 (94)	247.46	396.47	-45.39%
scpb3-3	9324	**4028**	4563.00	3396 (104)	126.55	262.64	-56.80%	5864	**2609**	2851.50	1776 (77)	302.82	446.05	-55.51%
scpb4-3	9576	**4279**	4642.00	3564 (105)	47.21	276.72	-55.32%	5145	**2811**	3064.50	1824 (81)	339.11	500.77	-45.36%
scpb5-3	10922	**4768**	5175.00	3682 (89)	51.97	294.25	-56.34%	6178	**2933**	3421.00	2072 (74)	190.47	357.29	-52.53%
scpc1-3	17478	**7562**	8061.50	3600 (36)	554.82	-	-56.73%	1645	**1608**	1668.50	0 (0)	351.85	-	-2.25%
scpc2-3	7473	**4158**	4431.50	2541 (70)	341.92	-	-44.36%	1744	**1301**	1372.50	330 (10)	523.38	-	-25.40%
scpc3-3	10699	**4958**	5135.00	2200 (41)	354.67	-	-53.66%	1625	**1372**	1459.00	150 (3)	508.60	-	-15.57%
scpc4-3	7997	**4437**	4882.50	2584 (54)	365.07	-	-44.52%	1636	**1406**	1448.50	114 (3)	600.98	-	-14.06%
scpc5-3	10557	**5041**	5575.50	2250 (40)	434.66	-	-52.25%	1640	**1370**	1442.50	50 (1)	582.99	-	-16.46%
scpd1-3	12601	**5900**	6187.00	5436 (106)	144.55	524.91	-53.18%	8882	**4383**	4782.00	3456 (127)	415.66	503.97	-50.65%
scpd2-3	11682	**6017**	6164.50	5310 (130)	349.14	523.48	-48.49%	9258	**4268**	4825.00	3654 (123)	262.83	525.81	-53.90%
scpd3-3	11049	**6055**	6344.50	5373 (127)	403.17	525.33	-45.20%	9181	**4536**	4847.00	3312 (128)	162.69	569.60	-50.59%
scpd4-3	13535	**8213**	9022.00	6482 (130)	126.30	511.43	-39.32%	13738	**6393**	6938.50	5334 (123)	133.36	561.40	-53.46%
scpd5-3	13176	**6181**	6269.00	5580 (145)	407.39	590.85	-53.09%	8403	**4704**	4868.00	3618 (136)	53.63	486.80	-44.02%
scpe1-3	26*	**26**	26.00	17 (3)	3.55	12.34	0.00%	23*	**23**	23.00	14 (3)	0.39	12.11	0.00%
scpe2-3	28*	**28**	28.00	19 (3)	0.19	12.50	0.00%	25*	**25**	25.00	16 (3)	0.62	11.88	0.00%
scpe3-3	24*	**24**	24.00	18 (1)	0.06	10.83	0.00%	23*	**23**	23.00	17 (1)	0.03	10.28	0.00%
scpe4-3	25*	**25**	25.00	16 (3)	0.81	12.22	0.00%	22*	**22**	22.00	13 (3)	2.08	12.48	0.00%
scpe5-3	28*	**28**	28.00	19 (3)	1.02	12.42	0.00%	25*	**25**	25.00	16 (3)	0.70	11.56	0.00%
Set-B														
scpclr10-3	1926*	**1926**	1926.00	1893 (55)	1.75	24.59	0.00%	1871*	**1871**	1871.00	1838 (55)	0.39	24.75	0.00%
scpclr11-3	4215	**3501**	3501.00	3468 (55)	0.25	30.97	-16.94%	3890	**3446**	3446.00	3413 (55)	4.34	31.11	-11.41%
scpclr12-3	10143	**6429**	6429.00	6396 (55)	6.36	45.56	-36.62%	7728	**6374**	6374.00	6341 (55)	21.97	45.61	-17.52%
scpclr13-3	25138	**13267**	13677.50	13234 (55)	5.03	81.02	-47.22%	29272	**13212**	14033.00	13179 (55)	8.98	85.83	-54.86%
scpcyc06-3	126*	**126**	126.00	42 (27)	0.30	14.02	0.00%	99*	**99**	99.00	15 (9)	0.47	15.60	0.00%
scpcyc07-3	335*	**335**	335.00	136 (74)	3.81	43.02	0.00%	250*	**250**	251.00	49 (38)	28.31	51.17	0.00%
scpcyc08-3	911	**908**	911.00	434 (192)	57.19	463.93	-0.33%	**662**	666	688.00	180 (154)	370.07	596.71	0.60%
scpcyc09-3	2332	**2320**	2328.50	1228 (487)	86.30	-	-0.51%	**1678**	1741	1758.50	607 (474)	172.80	-	3.75%
scpcyc10-3	6755	**5718**	5766.50	3270 (1234)	736.04	-	-15.35%	**4441**	4486	4539.00	1900 (1426)	41.88	-	1.01%
scpcyc11-3	19422	**13668**	13905.50	8265 (2979)	858.34	-	-29.63%	12771	**11140**	11308.00	5347 (3885)	338.55	-	-12.77%
Set-C														
scpnre1-3	7424	**3987**	4095.00	3447 (36)	552.57	-	-46.30%	7047	**3858**	3970.50	3303 (36)	20.72	578.23	-45.25%
scpnre2-3	9054	**3863**	4010.50	3603 (36)	76.56	597.94	-57.33%	8820	**3768**	3964.00	3447 (36)	429.05	-	-57.28%
scpnre3-3	8211	**3888**	4017.50	3288 (36)	4.61	579.29	-52.65%	6581	**3714**	3916.50	3207 (36)	135.05	-	-43.56%
scpnre4-3	7230	**3891**	4089.00	3360 (36)	40.36	593.94	-46.18%	6327	**3713**	3915.00	3144 (36)	191.94	-	-41.31%
scpnre5-3	7488	**3870**	4115.00	3615 (36)	12.86	568.54	-48.32%	7323	**3685**	3906.00	3456 (36)	220.39	-	-49.68%
scpnrf1-3	2145	**1185**	1215.00	1098 (10)	378.10	-	-44.76%	2798	**1175**	1191.00	1088 (10)	193.77	-	-58.01%
scpnrf2-3	2324	**1220**	1243.00	1124 (10)	149.39	-	-47.50%	2276	**1210**	1232.00	1114 (10)	230.88	-	-46.84%
scpnrf3-3	2241	**1184**	1205.00	1088 (10)	476.99	-	-47.17%	2228	**1186**	1195.00	1048 (10)	144.09	-	-46.77%
scpnrf4-3	2484	**1183**	1230.00	1105 (10)	10.83	-	-52.38%	2440	**1173**	1225.50	1095 (10)	341.07	-	-51.93%
scpnrf5-3	2499	**1226**	1230.00	1138 (10)	388.18	-	-50.94%	2327	**1216**	1231.00	1128 (10)	55.44	-	-47.74%
scpnrg1-3	33276	**15011**	15434.00	12650 (671)	595.02	-	-54.89%	21735	**6657**	6941.00	4230 (282)	519.77	-	-69.37%
scpnrg2-3	34828	**14731**	15155.50	12180 (600)	239.76	-	-57.70%	15524	**6963**	7356.50	4520 (317)	579.98	-	-55.15%
scpnrg3-3	28168	**16381**	16891.50	12859 (624)	354.27	-	-41.85%	17639	**7574**	7961.00	5522 (311)	492.17	-	-57.06%
scpnrg4-3	30665	**14860**	15580.00	11710 (619)	281.20	-	-51.54%	17604	**7145**	7600.50	5010 (306)	248.52	-	-59.41%
scpnrg5-3	26931	**14862**	15442.50	11570 (603)	67.70	-	-44.81%	15825	**7255**	7672.50	5070 (329)	527.78	-	-54.15%
scpnrh1-3	19313	**10252**	10560.50	9363 (171)	241.14	-	-46.92%	20124	**9516**	9810.50	8187 (210)	307.50	-	-52.71%
scpnrh2-3	18537	**10342**	10482.00	9243 (190)	179.18	-	-44.21%	19527	**9654**	9899.50	8742 (190)	70.70	-	-50.56%
scpnrh3-3	17982	**9996**	10495.50	8775 (210)	161.74	-	-44.41%	18873	**9635**	9907.00	8796 (190)	116.36	-	-48.95%
scpnrh4-3	16749	**10341**	10568.00	9315 (190)	123.13	-	-38.26%	16815	**9036**	9975.50	8346 (210)	306.42	-	-46.26%
scpnrh5-3	17361	**10291**	10487.00	9417 (190)	61.38	-	-40.72%	16461	**9309**	9736.50	8352 (171)	213.13	-	-43.45%
Avg						392.73	-27.90%						376.33	-22.71%
#Best/Opt		75							76					
AvgRSD		1.65%							1.22%					

Table 5.4: Gurobi vs. parallel GRASP: computational results

Chapter 6

Max Flow with Conflicts

Maximum Flow Problems (MFPs) have been extensively studied over the years by researchers in optimization theory, and were used to support real-world problems related to transportation and communications. In this chapter, we face a variant of the MFP problem in which a list of conflicting edge pairs is available and the maximum flow is computed by assuring that there are no edges in conflict used to carry this flow.
To the best of our knowledge, Pferschy et al. [PS13b] have introduced the variant of the maximum flow problem by considering positive and negative disjunctive constraints and they prove the NP-hardness of these variants. The variant of our interest is the one with negative disjunctive constraints, which are the constraints used to model conflicts between arc pairs, named Maximum Flow with Conflicts constraints (MFPC). Suvak et al. [cAA20] proposed several mixed-integer linear programming formulations for MFPC.
MFPC consists in determining the maximum flow that can be sent on a flow network from a source node to a sink node respecting the classical capacity constraints over the arcs (already present in the original problem) and the conflict constraints, that forbid to carry flow through two arcs in conflict. To face the problem, we propose some heuristic approaches. Firstly, we introduce a modification of the well-known augmenting-paths algorithm, commonly used to solve the classic MFP, which is based on a greedy criterion, that quickly allows to identify a feasible solution in presence of conflicting arcs. Then, we present a more accurate algorithm, obtained by embedding the introduced greedy algorithm into a Carousel Greedy (CG) framework. To further address the problem, we design an algorithm based on the Kernel Search (KS) scheme, which contains inside our CG approach; we called this approach Kernousel. The readers can find a description of CG and KS methodologies in Sections A.2.1 and A.5.1 of Appendix A, respectively.

6.1 Problem description and formulations

The Maximum Flow problem with additional conflict constraints (MFPC) is defined on a directed graph $G = (V, A)$, with V denoting the set of vertices and A denoting the set of directed arcs of

G. Along with the given network, a source vertex s and a sink vertex t are indicated. A function $u : A \to \mathbb{N} \cup \{0\}$ is provided, with u_{ij} indicating the maximum allowed flow on the arc $(i, j) \in A$. Two arcs $(i, j), (k, \ell) \in A$, $(i, j) \neq (k, \ell)$, are *conflicting* if, in any feasible solution of MFPC, at most one of them can have positive flow. Let $\delta(i, j)$ be the set of arcs conflicting with the arc $(i, j) \in A$. The aim of the problem is to identify the maximum flow from s to t without violating any conflict restriction.

6.2 Mathematical Models

The Kernel Search described in Section 6.3 uses two Mixed-Integer Linear Programming formulations of MFPC, proposed in [cAA20], named MFPC_s and MFPC_w, respectively. Since the polyhedron of MFPC_s is contained in that of MFPC_w, its LP relaxation produces better bounds and then it is referred as *strong formulation*, while MFPC_w is named *weak formulation*.

Formulation 3. *MFPC$_s$ (Strong Formulation)*

$$\max v \tag{6.1a}$$

$$s.t. \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = \begin{cases} v & \text{if } i = s \\ 0 & \text{if } i \in V \setminus \{s, t\} \\ -v & \text{if } i = t \end{cases} \qquad \forall\, i \in V \tag{6.1b}$$

$$x_{ij} + x_{k\ell} \leq 1 \qquad \forall\, (i, j) \in A,\ \forall\, (k, \ell) \in \delta(i, j) \tag{6.1c}$$

$$0 \leq f_{ij} \leq u_{ij} x_{ij} \qquad \forall\, (i, j) \in A \tag{6.1d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in A. \tag{6.1e}$$

Formulation 4. *MFPC$_w$ (Weak Formulation)*

$$\max v \tag{6.2a}$$

$$s.t. \sum_{(i,j)\in A} f_{ij} - \sum_{(j,i)\in A} f_{ji} = \begin{cases} v & \text{if } i = s \\ 0 & \text{if } i \in V \setminus \{s, t\} \\ -v & \text{if } i = t \end{cases} \qquad \forall\, i \in V \tag{6.2b}$$

$$|\delta(i, j)| x_{ij} + \sum_{(k,\ell)\in\delta(i,j)} x_{k\ell} \leq |\delta(i, j)| \qquad \forall\, (i, j) \in A \tag{6.2c}$$

$$0 \leq f_{ij} \leq u_{ij} x_{ij} \qquad \forall\, (i, j) \in A \tag{6.2d}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, (i, j) \in A. \tag{6.2e}$$

Both Formulations 3 and 4 rely on the linear decision variables $f \in \mathbb{R}_0^{+|A|}$, indicating the flow assigned to each arc, the linear auxiliary variable v, representing the overall flow from s to t, and the binary auxiliary variables $x \in \{0, 1\}^{|A|}$, equal to 1 if and only if the related arcs carry some flow. Furthermore, both the formulations make use of the classical flow-balance equality

constraints (6.1b) and (6.2b), as well as the classical flow capacity constraints, modified in order to correctly set the values of the auxiliary variables x to 1 if the flow on the related arcs is positive, (6.1d) and (6.2d) respectively.

The two formulations only differ in how they prevent conflict violations. In particular, constraints (6.1c) guarantee, for each pair of conflicting arcs (i, j) and (k, ℓ), that at most one of them has positive flow, while constraints (6.2c) impose that every arc $(i, j) \in A$ has flow equal to zero if positive flow is assigned to any of the arcs conflicting with (i, j), i.e., the arcs in $\delta(i, j)$.

In the following, we denote by LP-MFPC$_\text{s}$ and LP-MFPC$_\text{w}$ the linear relaxations of the strong and weak formulations, respectively. Given a subset of arcs $F \subseteq A$, we further denote by MFPC$_\text{s}(F)$ and MFPC$_\text{w}(F)$ the strong and the weak formulations, in which the set of variables is restricted to those associated with the arcs in F, while LP-MFPC$_\text{s}(F)$ and LP-MFPC$_\text{w}(F)$ are their linear relaxations, respectively.

6.3 Heuristic approaches for MFPC

In this section, a detailed description of the heuristic and matheuristic methods designed to address the MFPC is given. In particular, Subsection 6.3.1 introduces an adaptation of the well-known augmenting path algorithm for the MFPC, while Subsection 6.3.2 describes an enhanced algorithm, obtained by embedding the greedy algorithm described in Subsection 6.3.1 in the Carousel Greedy (CG) framework. In Subsection 6.3.3, a Kernel Search (KS) based algorithm is described. Finally, in Subsection 6.3.4, the novel *Kernousel* approach, consisting of a KS algorithm which incorporates the previously described CG algorithm, is presented.

6.3.1 Greedy algorithm

The *greedy* algorithm described in this section is a heuristic adaptation of the classic augmenting path algorithm used to solve the classical maximum flow problem. The key restrictions of such adaptation concern *(i)* the update of the residual network after a given number of units flows along a path P, which implies the exclusion of all the arcs in conflict with the ones traversed by P; and *(ii)* the ad-hoc strategy according to which every augmenting path is detected, that prevents any violation with respect to the given set of conflicts. The algorithm makes use of the well-known capacity scaling approach, in order to find the augmenting paths with highest capacity first, motivated by the aim of faster obtaining higher flow values.

Formally, given a capacitated network $G = (V, A)$ and a flow $f = \{f_{ij}\}$, i.e., a vector satisfying constraints (6.1b) and (6.1d), the *residual capacity* associated with any pair of vertices $(i, j) \in V \times V$, with respect to f, is denoted by $r_{ij}^{(f)} = u_{ij} - f_{ij} + f_{ji}$. Furthermore, the *residual network* of G induced by f is defined as $G_f = (V_f, A_f)$, where $V_f = V$ and $A_f = \{(i, j) \in V \times V : r_{ij}^{(f)} > 0\}$. Any directed s-t path in G_f is an augmenting path.

According to the given definition, A_f potentially includes arcs conflicting with some of the arcs used by f, i.e., arcs $(i, j) \in A$ s.t. $f_{ij} > 0$, as well as further pairs of arcs in conflict with each other. Selecting any augmenting path using one of those arcs/pairs would result in

Algorithm 12: MFPC-Greedy

Data: Original graph G; source vertex s; sink vertex t.
Result: Ordered sequence of the identified conflict-free augmenting paths $S = \langle P_k \rangle_{k \in \{1,\dots,|S|\}}$; augmenting paths capacities $\Delta = \langle \Delta_{P_k} \rangle_{k \in \{1,\dots,|S|\}}$; final residual graph G_f.

1 $S \leftarrow \langle\rangle$; $\Delta \leftarrow \langle\rangle$;
2 $f_{ij} \leftarrow 0$, $\forall\, (i,j) \in V \times V$;
3 $G_f \leftarrow$ residual graph of G w.r.t. f;
4 **while** ComputeNextPath identifies an augmenting path **do**
5 $\quad (P, \Delta_P) \leftarrow \text{ComputeNextPath}(G, G_f, s, t, \langle\rangle, None)$;
6 $\quad$ augment Δ_P units of flow along P and update f and G_f, accordingly;
7 $\quad$ remove from G_f all the arcs in $\bigcup_{(i,j) \in A_f : f_{(i,j)} > 0} \delta(i,j)$;
8 $\quad$ add P to S and Δ_P to Δ;
9 **end**
10 **return** S, Δ, G_f

violating the conflict constraints of the problem. Such violation is prevented in two stages. First, whenever G_f is updated, all the arcs conflicting with some of the arcs in $\bigcup_{(i,j) \in A_f : f_{(i,j)} > 0} \delta(i,j)$ are removed from A_f. Second, the ComputeNextPath procedure is left to identify a non-conflicting augmenting path in the resulting residual network G_f.

By iterating these two stages until an additional augmenting path is detected, the designed greedy algorithm incrementally builds an ordered sequence $S = \langle P_1, P_2, \dots, P_{|S|} \rangle$ of non-conflicting augmenting paths, along with a vector Δ, indexed by the identified paths, such that Δ_{P_k} is the augmenting capacity associated with P_k, $k \in \{1, \dots, |S|\}$.

The pseudocode of the high-level procedure is reported in Algorithm 12, which takes as input a graph $G = (V, A)$, a source vertex $s \in V$ and a sink vertex $t \in V$ such that $s \neq t$. In the initialization phase, the empty sequences S and Δ, as well as the zero-flow f, are initialized, and the auxiliary graph G_f is built (lines 1-3).

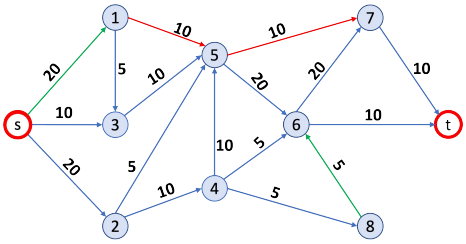
An iteration of the main loop (lines 4-9) is performed until the ComputeNextPath procedure detect an augmenting path, that does not violate any conflict, in the residual graph G_f. In each iteration, such procedure is invoked (line 5), generating a non-conflicting augmenting path P, as described in Algorithm 13. Then, the amount of flow Δ_P is sent along P and the flow and the residual network are updated accordingly (line 6). Subsequently, all the arcs that conflict with some of the arcs such that $f_{ij} > 0$ are removed from the residual network (line 7), while S and Δ are updated (line 8).

At the end of the k-th iteration, S stores exactly k conflict-free augmenting paths, according to the order in which they have been generated by the algorithm, i.e., $S = \langle P_i \rangle_{i \in \{1,\dots,k\}}$, while $\Delta = \{\Delta_{P_i}\}_{i=1,\dots,k}$ reports the associated augmenting capacities. In the style of a constructive algorithm, once a path is added to the solution, it is no longer modified or removed. How-

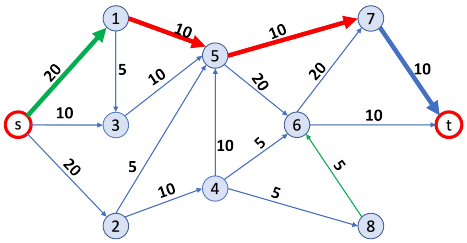
ever, it is worth noting that updating the residual network can still possibly make previously excluded arc(s) available again. This happens when the conflicting arc(s) used in the solution no longer carry flow. As a trivial example, consider $S = \langle P_1, P_2 \rangle$, where $P_1 = \langle (s,i),(i,j),(j,t) \rangle$, $P_2 = \langle (s,j),(j,i),(i,t) \rangle$ and $\Delta_{P_1} = \Delta_{P_2}$. It is easy to see that the actual flow f_{ij} carried by the arc (i,j) is Δ_{P_1} after the first iteration, and zero after the second one. Accordingly, when updating G_f after the first iteration, all the arcs in $\delta(i,j)$ are removed from A_f, while each of them is re-added after the second iteration, unless it is in conflict with some other arc of P_1 or P_2 with associated positive flow.

The ComputeNextPath procedure, used in line 5 of Algorithm 12, applies a greedy strategy in order to identify an augmenting path satisfying all the conflict constraints in G_f which locally seems to be the most preferable, i.e., the one with the highest associated augmenting capacity. The well-known capacity scaling approach [CLRS22] is adopted to compute the first s-t augmenting path, and a series of recursive iterations is performed if the produced path contains conflicting arcs. In this case, indeed, part of the initial path is stored and the source vertex is progressively moved to the tail of one of the conflicting arcs, referred to as *pivot* arc, in the following. The s-t path is then completed with different arcs and the whole process is iterated until a conflict-free path is obtained.

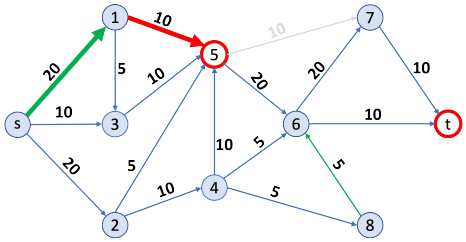
Figure 6.1 provides an example of the generation process. In particular, Figure 6.1a shows a flow graph with conflicting arc pairs, which cannot be used on the same path, namely $\{(s,1),(8,6)\}$ and $\{(1,5),(5,7)\}$. Graphically, the arcs involved in the same conflict are depicted using the same color, while the source and the sink vertices s and t are circled in red. Figure 6.1b shows in bold the first s-t path of high capacity computed by ComputeNextPath by temporarily ignoring the conflict constraints, i.e., $\langle (s,1),(1,5),(5,7),(7,t) \rangle$. Once computed, the algorithm checks if both arcs of any conflicting pair are in the path by scanning it from the source to the sink. If such pair is found, one of the two arcs is randomly chosen to stay and the other one is removed from the path. In the example, conflict $\{(s,1),(8,6)\}$ is not violated, because only $(s,1)$ belongs to the identified path, while $\{(1,5),(5,7)\}$ is violated, since both the arcs are traversed. Among the two arcs, $(1,5)$ is chosen to stay and $(5,7)$ is removed, as shown in Figure 6.1c. Subsequently, a new path is computed from vertex 5, i.e., the tail of the removed arc, to the sink. The obtained path, i.e., $\langle (5,6),(6,t) \rangle$, is shown in Figure 6.1d. Finally, the conflict check is repeated with respect to the whole resulting path, i.e., $\langle (s,1),(1,5),(5,6),(6,t) \rangle$. As no conflict violation is detected, the procedure stops, after returning the computed path.



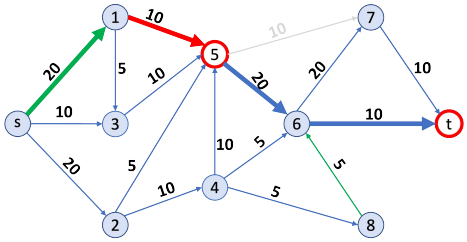
(a) Example of flow graph with conflicts. The vertices s and t are the source and sink of the graph, respectively. Arcs with the same color are conflicting.



(b) Augmenting path from s to t containing conflicting arcs



(c) Partial path obtained from 6.1b by discarding $(5, 7)$ and subsequent arcs. Vertex 5 is the new source.



(d) Complete path from s to t which does not contain conflicting arcs

Figure 6.1: Identification of an augmenting path without conflicts by the greedy algorithm.

The pseudocode of the ComputeNextPath procedure is reported in Algorithm 13. It receives as input the residual graph G_f, the source and the sink vertices s and t, together with two optional parameters: a partial solution P_{INIT} and a pivot arc (p, q), i.e., an arc in the outgoing star of the last vertex of P_{INIT} in G_f which must not belong to the identified path. If any partial solution is provided, i.e., P_{INIT} is not empty, the pivot arc is required. The computed path P, together with the associated amount of flow Δ_P, constitutes the output of the algorithm if such path is found. Otherwise, the algorithm does not return any solution. ComputeNextPath is a recursive procedure. In the very first call of the recursion process, no partial solution nor pivot arc are provided. The algorithm searches for an augmenting path P in G_f and then validates it with respect to the set conflicting arcs. If a violation is detected, ComputeNextPath is recursively invoked by specifying both a conflict-free portion of P and the first arc (p, q), which causes some conflict violation, as pivot. The path is incrementally completed through a series of recursion steps, until no violation is detected.

More in detail, the first operation performed by the algorithm is to check whether a partial path P_{INIT} is provided (line 1). If this is the case, the residual capacities of the pivot arc (p, q) and those of the arcs belonging to P_{INIT}, as well as the opposite arcs, are stored and temporarily set to zero in G_f, in order to exclude them from the computation of the new path (lines 2-3). The original residual capacities are afterwards restored on line 10. In order to compute an augment-

Algorithm 13: ComputeNextPath

Data: Residual graph G_f; source vertex s; sink vertex t; possibly empty partial solution path P_{INIT}; pivot arc (p, q), required in case a non-empty P_{INIT} is provided.
Result: Conflict-free s-t path P with the associated capacity Δ_P, if such path is found. $None$ otherwise.

1 **if** $|P_{INIT}| > 0$ **then**
2 $\quad \hat{r} \leftarrow$ current vector of the residual capacities of the arcs in G_f;
3 $\quad$ set $r_{pq} = 0$ and $r_{ij} = r_{ji} = 0$ in G_f, $\forall\ (i, j) \in P_{INIT}$;
4 **end**
5 $(\hat{P}, \Delta_{\hat{P}}) \leftarrow$ CapacityScaling(G_f, s, t);
6 **if** $\hat{P} = None$ **or** $\Delta_{\hat{P}} = 0$ **then return** $None$;
7 **else**
8 $\quad (P, s_N, (p_N, q_N)) \leftarrow$ ValidatePath$(\hat{P}, P_{INIT})$;
9 $\quad$ **if** $p_N \neq None$ **then** $(P, \Delta_P) \leftarrow$ ComputeNextPath$(G_f, s_N, t, P, (p_N, q_N))$;
10 $\quad$ set $r_{pq} = \hat{r}_{pq}$, $r_{ij} = \hat{r}_{ij}$ and $r_{ji} = \hat{r}_{ji}$ in G_f, $\forall\ (i, j) \in P_{INIT}$;
11 $\quad$ **return** P, Δ_P;
12 **end**

ing s-t path, the well-known capacity scaling approach [CLRS22] is adopted (line 5), consisting in imposing a limit on the minimum amount of flow sent along the desired path and iteratively decreasing such limit until a path is found. If it is not possible to send additional flow from s to t on G_f, the path $\hat{P}$ and the flow Δ_P placeholders are assigned no value. In this case, the ComputeNextPath procedure stops and returns $None$ (line 6). Otherwise, in order to check for conflicting arcs, potentially included in $\hat{P}$, the ValidatePath procedure is exploited (line 8). If $\hat{P}$ contains two conflicting arcs, (p_N, q_N), s_N and P are the selected pivot arc, the new source vertex, namely the tail of the pivot arc, and the partial conflict-free solution path P, which contains all the arcs of the previous partial path P_{INIT} and all the arcs preceding p_N in $\hat{P}$, respectively. In order to obtain a sub-path from the new source s_N to the sink t, to be concatenated to P, ComputeNextPath is recursively invoked (line 9). Otherwise, if no conflict violation is detected in $\hat{P}$, no value is assigned to (p_N, q_N) and s_N, while P becomes the concatenation of P_{INIT} and $\hat{P}$ and is in the end returned (line 11).

Finally, the ValidatePath procedure, whose pseudocode is reported in Algorithm 14, is used to validate the generated augmenting paths, with respect to the defined set of conflicts. The algorithm takes as input two paths P_{sk} and P_{kt}: the former is a partial path going from the source vertex s to some intermediate vertex $k \neq t$, which does not contain any conflicting arc pairs and can possibly be empty; the latter, instead, is a candidate completing path going from vertex k to the target vertex t and potentially violates conflicts. Clearly, if P_{sk} is empty, $k = s$, i.e., P_{kt} starts from the source vertex s. The output of the algorithm consists of a conflict-free

Algorithm 14: ValidatePath

Data: Conflict-free partial path P_{sk}; candidate completing path P_{kt}.
Result: Conflict-free s-t solution path P if no conflict is detected in $\langle P_{sk}, P_{kt} \rangle$;
conflict-free partial path P with a new pivot arc (p_N, q_N) and a new source s_N, otherwise.

1 $P \leftarrow P_{sk}$; $s_N \leftarrow None$; $(p_N, q_N) \leftarrow None$;
2 **for** (i, j) *in* P_{kt} **do**
3 **if** $\delta(i, j) \cap P = \emptyset$ **then** P.append((i, j));
4 **else**
5 **if** $random(0, 1) = 0$ **then** $(p_N, q_N) \leftarrow (i, j)$; $s_N \leftarrow i$;
6 **else**
7 $(k, l) \leftarrow$ first arc in $\delta(i, j) \cap P$;
8 $P \leftarrow \langle (p, q) \in P : (p, q)$ precedes $(k, l) \rangle$;
9 $(p_N, q_N) \leftarrow (k, l)$; $s_N \leftarrow k$;
10 **end**
11 **return** $P, s_N, (p_N, q_N)$;
12 **end**
13 **end**
14 **return** $P, s_N, (p_N, q_N)$;

path P, which is a s-t path if no conflicting arc pair has been detected among all the arcs of P_{sk} and P_{kt}. If any conflict is detected, P is a partial path, instead, and a pivot arc (p_N, q_N), i.e., the first excluded arc of P_{kt}, along with its tail vertex s_N, are returned.

In the initialization phase, P is assigned the partial path P_{sk}, while (p_N, q_N) and s_N assume no value (line 1). Then, the arcs in the candidate path P_{kt} are processed in the order they appear, until a violated conflict has been detected or the whole path has been analysed (lines 2-13). In particular, each arc $(i, j) \in P_{kt}$ is added to P if such insertion does not compromise the feasibility of P, i.e., if (i, j) does not conflict with any of the arcs currently in P (line 3). Otherwise, with a half probability, (i, j) becomes the first excluded arc of P_{kt}, i.e., $(p_N, q_N) = (i, j)$ and $s_N = i$; with the same probability, the choice falls on the first arc $(k, l) \in \delta(i, j) \cap P$, namely the first arc of P conflicting with (i, j), and all the arcs placed after (k, l) are removed from P (lines 5-10); in both cases, the execution stops and the extended partial path is returned, along with the selected pivot arc and its tail (line 11). If no conflicting arcs are detected in any iteration, at the end of the execution P is a conflict-free s-t path, coinciding with the concatenation of P_{sk} and P_{kt}. Such complete path is returned along with the unassigned s_N and (p_N, q_N) (line 14).

6.3.2 Carousel Greedy

In the following, we provide a detailed description of the CG algorithm that we designed for the MFPC, which incorporates the greedy procedure introduced in Subsection 6.3.1.

The CG framework allows designing enhanced greedy algorithms, which generally exhibit higher accuracy than the underlying greedy algorithms, with very reduced computational overhead. More in detail, provided a starting solution of a greedy algorithm, a noteworthy improvement is obtained by iteratively discarding the oldest choices made by the algorithm and by performing some new choices. By doing so, the very final phase of the algorithm, where choices are made taking into account significant information gained throughout the computation, is extended, as opposite to what happens at the beginning of the computation, when few information is available and the performed choices have higher impact on the future ones.

Designing a CG algorithm involves specifying the values of two parameters, namely the number of iterations to be performed w.r.t. the starting solution size (α) and the percentage of the items, composing the tail of the starting solution, to be removed at the beginning of the computation (β).

The scheme shown in Figure 6.2 graphically visualizes the main idea behind the CG algorithm designed for the MFPC. The whole process starts from obtaining a feasible solution by executing the MFPC-Greedy algorithm. As described in Section 6.3.1, such solution is composed of a sequence S of non-conflicting augmenting paths, i.e., $S = \langle P_1, P_2, \ldots, P_{|S|} \rangle$, appearing in the order they have been selected by the greedy algorithm. As long as the CG algorithm is executed, such sequence is progressively updated. Firstly, the last $\beta|S|$ augmenting paths are removed from S, obtaining the so-called *carousel start*. Then, the $\alpha|S|$ iterations of the CG algorithm are performed. In each iteration, the left-most item, i.e., the oldest selected augmenting path, is discarded and the ComputeNextPath procedure is exploited to obtain another conflict-free augmenting path, if any, with respect to the updated S. After the last iteration, the resulting collection of non-conflicting augmenting paths S is completed by performing the MFPC-Greedy procedure.

Unlike the classical CG approach, introduced in [CCG17], where a feasible solution is obtained only at the end of the computation, the designed CG algorithm for the MFPC, by construction, yields a feasible solution at the end of each iteration. Additionally, to further intensify the exploration of the solution space, in the designed CG algorithm, whenever an augmenting path is removed from the current solution, its first arc is temporarily assigned zero capacity in the residual network, thus preventing the greedy procedure from selecting the just excluded path in the same iteration. In the same vein, a frequency vector storing the occurrences of each generated path is maintained and a capacity equal to zero is temporarily assigned to the arcs of the most chosen path. Nonetheless, the MFPC-Greedy algorithm is run at the end of each iteration, thus obtaining an intermediate complete solution, i.e., a collection of paths that the algorithm is no longer able to extend with additional augmenting paths. Although the paths generated by the algorithm to obtain such solution are not added to S, the value of the obtained solution is still compared with the incumbent solution, resulting in a possible update of the best solution found. Contextually, a collection $\mathcal{P}$ of all the paths computed by the algorithm is built and returned at the end of the computation.

The pseudocode of the designed CG algorithm is reported in Algorithm 15, which takes as

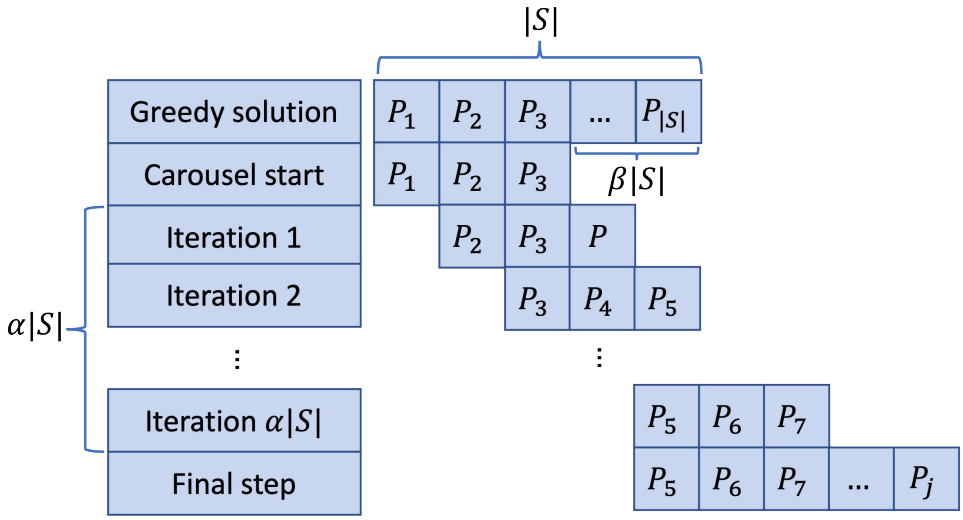


Figure 6.2: Scheme of the Carousel Greedy algorithm designed for the MFPC.

input the original graph G, the source vertex s, the sink vertex t and the α and β parameters. At the end of the computation, the output of the algorithm consists of the best found solution S^*, i.e., an ordered list of conflict-free augmenting paths yielding the highest known flow, along with the collection $\mathcal{P}$ of all the generated augmenting path and a frequency vector τ, where τ_P indicates the number of times that path P has been considered by the algorithm, for each $P \in \mathcal{P}$.

In the initialization phase, the MFPC-Greedy procedure is used to obtain a starting initial solution $S = \langle P_1, \ldots, P_{|S|} \rangle$, along with the associated flow vector Δ and residual graph G_f (line 1). Then, the best solution found S^*, the related flow value z^*, the collection of generated paths $\mathcal{P}$ and the frequency vector τ are initialized, accordingly (lines 2-3). At this point, τ indicates that each path of the starting solution has been chosen exactly once. Subsequently, the so-called *carousel start* S^C is obtained, by removing from S the last $\beta\%$ of its paths (line 4), and the $\alpha|S|$ CG iterations are performed (lines 5-20).

At the beginning of each iteration, the oldest path chosen according to the greedy strategy, denoted by P^O, is removed from S^C (line 6), then the residual graph G_f associated to the flow induced by the resulting solution S^C is computed and all the arcs conflicting with some of the used arcs are removed from G_f (lines 7-8); subsequently, the first arc of P^O and all the arcs belonging to the most frequently chosen path P_τ are set to zero (lines 9-10), which guarantees that the next selected path is different from the just removed one, encouraging the exploration of the solution space. At this point, the ComputeNextPath procedure is used to identify the next most promising non-conflicting path (line 11). If any augmenting path P is found, it is added to the collection $\mathcal{P}$ and the number τ_P of times P has been chosen is updated, as well as the current solution S^C, the current flow f and the resulting residual graph G_f. Furthermore,

Algorithm 15: MFPC-CarouselGreedy

Data: Original graph G; source vertex s; sink vertex t; α and β parameters.
Result: Best found solution S^*; collection $\mathcal{P}$ of all the augmenting paths generated during the computation and frequency vector τ.

1 $(S, \Delta, G_f) \leftarrow$ MFPC-Greedy(G, s, t);
2 $S^* \leftarrow S$; $z^* \leftarrow f(S)$;
3 $\mathcal{P} \leftarrow \{P_i : P_i \in S\}$; $\tau_{P_i} \leftarrow 1$, $\forall\ P_i \in S$;
4 initialize the carousel start S^C with the first $\lfloor (1-\beta)|S| \rfloor$ paths in S;
5 **for** $i \in \{1, \ldots, \alpha|S|\}$ **do**
6 remove the oldest path P^O from S^C;
7 compute the residual network G_f of G w.r.t. the flow induced by S^C;
8 remove from G_f all the arcs in conflict with some of the arcs of any path in S^C;
9 let P_τ be the path in $\mathcal{P}$ with highest frequency, i.e., $P_\tau := \arg\max_{P \in \mathcal{P}} \{\tau_P\}$;
10 set $r_{ij} = r_{ji} = 0$ in G_f the first arc of P^O and all the arcs in P_τ;
11 $(P, \Delta_P) \leftarrow$ ComputeNextPath$(G, G_f, s, t, \langle\rangle, None)$
12 **if** $P \neq None$ **and** $\Delta_P > 0$ **then**
13 $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$; $\tau_P \leftarrow \tau_P + 1$;
14 add P to S^C and update f and G_f, accordingly;
15 **if** $z^* \leq f(S^C)$ **then** $S^* \leftarrow S_C$; $z^* \leftarrow f(S^C)$;
16 **end**
17 obtain an intermediate complete solution S^{C+} by running MFPC-Greedy(G, s, t) with the starting partial solution $S = S^C$;
18 $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\}$, $\tau_P \leftarrow \tau_P + 1, \forall\ P \in S^{C+} \setminus S^C$;
19 **if** $z^* \leq f(S^{C+})$ **then** $S^* \leftarrow S^{C+}$; $z^* \leftarrow f(S^{C+})$;
20 **end**
21 **return** S^*, $\mathcal{P}$, τ;

if the flow value associated to the current solution is higher than the best known one, the best solution is updated (lines 12-16). Finally, the MFPC-Greedy algorithm is used to obtain an intermediate complete solution starting from S^C (line 17). According to the resulting solution, denoted by S^{C+}, the occurrences of each path and, possibly, the best found solution, are updated (lines 18-19). Note that, since S^C is not modified, this operation has no impact on the current solution, however it may influence the computation of the most frequently chosen path P_τ at the next iteration. In the end, the algorithm returns the best solution found S^*, the collection of generated paths $\mathcal{P}$ and the frequency vector τ.

6.3.3 Kernel Search

In this subsection, we describe the KS-based algorithm developed for the MFPC. As already mentioned, the founding idea of the KS framework is to identify a subset of promising items

which compose the *kernel* set (Λ), based on the information gained by solving the continuous relaxation of the problem. All the items initially excluded from Λ are instead organized in a collection of *buckets* $B = \langle B_1, \ldots, B_{|B|} \rangle$. The size of Λ, as well as the size of each $B_i \in B$, constitute the two key parameters of a KS-based algorithm. In the following, we will denote by λ and γ the kernel size and the bucket size, respectively. The choice of such parameters is crucial for the effectiveness and the performance of the resulting algorithm. Indeed, a series of mixed-integer linear programs, restricted to the variables associated with the items in Λ and one of the buckets at time, are solved by using state-of-the-art solvers. The larger the size of the restricted problems, the higher the quality of the final solution. Clearly, larger subproblems require more computing time to be solved. Lastly, a necessary condition for the applicability of KS is the meaningfulness of the solution provided by the continuous relaxation of the problem, w.r.t. the optimal integer solution.

The KS framework consists of two phases:

- **Initialization phase**, where the initial kernel is identified, along with the collection of buckets;
- **Extension phase**, where the kernel set is iteratively extended, by moving items from the buckets, whose inclusion improves the best known solution.

Figure 6.3 shows the content of such two phases w.r.t. the KS-based algorithm developed for the MFPC. In the initialization phase, the continuous relaxation LP-MFPC$_\text{s}$ is solved, thus obtaining a realization $(\underline{f}, \underline{x})$ of the flow and activation variables. Then, the arcs of G are sorted in non-increasing order by considering, for each arc $(i, j) \in A$, the value of the associated variable $\underline{f}_{ij}$, when $\underline{f}_{ij} > 0$, and its reduced cost coefficient, when $\underline{f}_{ij} = 0$. The first λ arcs, according to the resulting order, constitute the kernel set Λ, while the remaining ones are orderly partitioned into buckets of size γ. This process has been shown to generally produce, for the MFPC, a kernel set Λ with a reduced number of conflicting pairs. The first integer solution is then obtained by solving MFPC$_\text{w}(\Lambda)$, if the restricted problem is feasible. Nevertheless, in the extension phase, Λ is iteratively extended by solving several instances of MFPC$_\text{w}(\Lambda_i)$, where each superset $\Lambda_i \supseteq \Lambda$ is obtained at iteration i, by considering the bucket $B_i \in B$ and setting $\Lambda_i = \Lambda \cup B_i$. When solving the i-th subproblem, i.e., MFPC$_\text{w}(\Lambda_i)$, the following two additional constraints are imposed:

$$\sum_{(i,j) \in B_i} x_{ij} \geq 1 \tag{6.3}$$

$$v \geq w^*. \tag{6.4}$$

Constraint (6.3) requires that at least one of the activation variables whose associated arcs are in B_i assumes value 1 in the solution, while constraint (6.4) impose that the value of the solution identified at iteration i is higher than the one found at iteration $i - 1$. If the resulting problem is feasible, Λ is extendend by including every arc from B_i which has been assigned positive flow in the provided solution.

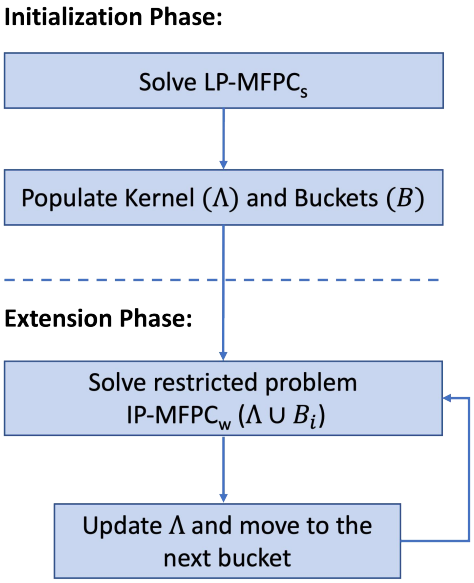


Figure 6.3: Kernel Search diagram.

6.3.4 Kernousel

In this section, we propose an hybrid heuristic algorithm for the MFPC, based on a novel resolution paradigm, named *Kernousel* (KO), which consists in enhancing a KS-based algorithm by exploiting a CG algorithm in the initialization phase.

The idea behind the proposed approach is to combine the optimality guarantee provided by the KS for restricted subproblems with the knowledge gained by the CG algorithm during its exploration of the solution space, including information about items which are not included in the final solution. We argue that, for highly constrained problems, where the inclusion of an item highly affects the rest of the solution, as it occurs for the MFPC, such paradigm may yield higher quality solutions than the singular approaches applied separately.

More in detail, in the Kernousel framework, all the paths generated during the execution of the CG algorithm, including the ones which do not actually belong to the final solution, are stored and then used to improve the composition of both the kernel set and the collection of buckets. Due to the given conflicting arc pairs, the discarded paths represent alternatives to the ones included in the solution provided by the CG and the arcs belonging to such paths are likely to be part of an optimal solution. Once the kernel set and the buckets have been initialized by considering the information collected by the CG algorithm, the remaining steps of the classical KS are performed.

The pseudocode of the resulting MFPC-Kernousel algorithm, which mostly reflects the high-

Algorithm 16: MFPC-Kernousel

Data: Graph G, source vertex s, sink vertex t, GC parameters α and β, kernel size λ and bucket size γ.

Result: Best found solution (f^*, x^*), along with its associated value w^*.

1 $S, \mathcal{P}, \tau \leftarrow$ MFPC-CarouselGreedy(G, s, t, α, β)

2 $\Lambda, B, w^* \leftarrow$ MFPC-Kernousel-Initialization$(G, s, t, \lambda, \gamma, \mathcal{P}, \tau)$

3 $f^*, x^*, w^* \leftarrow$ MFPC-Kernousel-Extension(Λ, B, w^*)

4 **return** f^*, x^*, w^*

level KS scheme shown in Figure 6.3, is reported in Algorithm 16, which takes as input the graph G, the source vertex s and the sink vertex t, as well as the CG parameters α and β, the kernel size λ and the bucket size γ. The output of the algorithm consists of the best found solution (f^*, x^*), along with its associated objective value w^*. Before performing the initialization phase, the CG algorithm is run, by obtaining an initial solution S, as well as the collection of analyzed paths $\mathcal{P}$ and their associated frequency τ (line 1). Then, the initialization phase, tailored for the Kernousel approach, is performed (line 2), thus composing the kernel set Λ and the collection B of buckets, on the basis of both the optimal solution of the continuous relaxation of the problem and the collection of paths generated by the CG algorithm. Finally, after performing the classical KS extension phase (line 3), the best found solution (f^*, x^*) and the associated objective value w^* are returned.

Algorithm 17 initializes the kernel set and the buckets collection. It takes as input the graph G, the source vertex s, the sink vertex t, the kernel size λ, the bucket size γ and the collection $\mathcal{P}$ of all the paths generated by the CG algorithm, along with their frequency vector τ. The output of the algorithm consists of the initialized kernel set Λ and buckets collection B, along with the best known objective value w^*, computed by solving the restricted MILP w.r.t. the variables included in the starting kernel set. After initializing the best known objective value w^* to $-\infty$, the continuous relaxation of the original problem is solved by using the strong formulation LP-MFPC$_s$, by obtaining the solution (f, x) and the reduced cost coefficients ρ associated to the out-of-base flow variables (lines 1-3). A sorted sequence L of the arcs (i, j) of G is then built, by concatenating the sequence of arcs with strictly positive flow, sorted by non-increasing order of f_{ij}, and the sequence of arcs with associated zero flow, sorted on the basis of non-increasing values of the associated reduced cost coefficients (lines 4-6). The kernel set Λ is then initialized with all the arcs appearing in any of the paths of the collection $\mathcal{P}$ stored by the CG algorithm (line 7). If they are not enough to populate the kernel, i.e., if the so-obtained number of arcs is less than λ, the remaining $\lambda - |\Lambda|$ arcs are taken from L, in the order they appear (lines 8-11). Then, the collection of buckets B is built, by considering γ arcs at time from the sequence L, until zero or strictly less than γ arcs are left; in the latter case, the remaining arcs are included in an additional bucket, smaller than the previous ones (line 12). Finally, the restricted MILP, w.r.t. the variables included in the starting kernel set, is solved by using the weak formulation

Algorithm 17: MFPC-Kernousel-Initialization

Data: Graph G, source vertex s, sink vertex t, kernel size λ, buckets size γ, CG paths collection $\mathcal{P}$ and frequency vector τ.

Result: Kernel set Λ and buckets collection B, along with the best known objective value w^*, i.e., the optimal solution of $\text{MFPC}_\text{w}(\Lambda)$.

1 $w^* \leftarrow -\infty$
2 $f, x \leftarrow$ optimal solution of LP-MFPC_s
3 $\rho_{ij} \leftarrow$ reduced cost coefficient of f_{ij}
4 $L_> \leftarrow$ sequence of the arcs $(i,j) \in A$ s.t. $f_{ij} > 0$, sorted according to non-increasing f_{ij} values
5 $L_= \leftarrow$ sequence of the arcs $(i,j) \in A$ s.t. $f_{ij} = 0$, sorted according to non-increasing ρ_{ij} values
6 $L \leftarrow L_> + L_=$
7 $\Lambda \leftarrow \{(i,j) \in P : P \in \mathcal{P}\}$
8 **if** $|\Lambda| < \lambda$ **then**
9 $\quad \Lambda \leftarrow \Lambda \cup \{L_1, \ldots, L_{\lambda - |\Lambda|}\}$
10 $\quad L \leftarrow \langle L_{\lambda-|\Lambda|+1}, \ldots, L_{|L|} \rangle$
11 **end**
12 $B \leftarrow$ collection of $\lceil |L|/\gamma \rceil$ buckets s.t. each B_i contains the next γ arcs from L
13 $(f^*, x^*) \leftarrow$ optimal solution of $\text{MFPC}_\text{w}(\Lambda)$
14 **if** (f^*, x^*) *is feasible* **then** $w^* \leftarrow$ objective value associated to (f^*, x^*)
15 **return** Λ, B, w^*

and the best known objective value is possibly updated, before returning the initialized sets and objective value (lines 13-15).

The pseudocode of the MFPC-Kernousel-Extension procedure, implementing the KS logic, is reported in Algorithm 18, which takes as input the kernel set Λ, the collection of buckets B and the starting objective value w^* obtained by solving the restricted MILP w.r.t. the variables added to Λ during the initialization phase. Each bucket $B_i \in B$ is processed a single time (line 1) and, with respect to each of them, the Λ_i set, containing all the variables of the previous kernel plus the variables of the i-th bucket, is considered (line 2). The associated restricted MILP is solved by using the weak formulation, i.e., $\text{MFPC}_\text{w}(\Lambda_i)$, with the additional requirements of constraints (6.3) and (6.4) (line 3). If the resulting problem is not feasible, no action is performed, otherwise it means that a solution with higher value than the best known one has been found, and the value of w^* is updated accordingly. Furthermore, the subset of variables of B_i associated with a positive flow in the solution are moved to the kernel set (lines 4-7). In the end, the best found solution variables (f^*, x^*) and objective value w^* are returned.

Algorithm 18: MFPC-Kernousel-Extension

Data: Kernel set Λ, bucket collection B and starting objective value w^*.
Result: Best known solution (f^*, x^*) and related objective value w^*.

for $B_i \in B$ **do**
  $\Lambda_i \leftarrow \Lambda \cup B_i$
  $(f^*, x^*) \leftarrow$ optimal solution of $\text{MFPC}_\text{w}(\Lambda_i)$ + constraints (6.3) and (6.4)
  if (f^*, x^*) *is feasible* **then**
    $w^* \leftarrow$ objective value associated to (f^*, x^*)
    $\bar{\Lambda}_i \leftarrow \{(i,j) \in B_i : f^*_{ij} > 0\}$
    $\Lambda \leftarrow \Lambda \cup \bar{\Lambda}_i$
  end
end
return f^*, x^*, w^*

6.4 Computational tests

The algorithms described in this work have been tested on the collection of benchmark instances provided by Şuvak et al. [cAA20]. By construction, each instance admits at least one non-zero flow feasible solution, and its size is described by three parameters: the number of nodes (n); the arc density ($p = \frac{m}{n(n-1)}$, where m denotes the number of edges); and the conflict density ($d = \frac{2w}{m(m-1)}$, where w denotes the number of conflicting arc pairs). The instance set is composed by 160 instances, two of which are generated for each combination of the n, p and d values, with $n \in \{40, 50, 60, 70, 80\}$, $p \in \{0.3, 0.4, 0.5, 0.6\}$ and $d \in \{0.3, 0.4, 0.5, 0.6\}$. Each instance is assigned an ID between 1 and 160. Furthermore, according to the number of nodes, instances 1-96 (with $n \in [40, 60]$) and instances 97-160 (with $n \in [70, 80]$) are classified as Small and Large instances, respectively.

In this section, we compare the results and performances of the CG, KS and MFPC-Kernousel approaches against the ones of the best performing approach proposed in [cAA20], i.e., a Benders Decomposition (BD) algorithm based on Formulation 4. Since the computational results of the BD algorithm, reported in [cAA20], refer to tests carried out by using a `Intel Xeon CPU E5-2687W 3.10 GHz` processor with 315.4 gigaflops, while we conducted the numerical experiments related to all the methods presented in this work using a `Intel(R) Core i9-9880H CPU @ 2.30GHz` processor with 432.22 gigaflops, we scale all the running times of BD by dividing them for the conversion ratio $\mu = \frac{432.22}{315.4} \approx 1.37$. Clearly, by applying this conversion, the one hour time limit, imposed by Şuvak et al., corresponds to 2627.7 seconds in our setting.

In our experiments, $\alpha = 40$, $\beta = 40$, $\lambda = 200$ and $\gamma = 250$.Compared with the typical values of α and β from the CG literature, the selected values are noticeably higher. The motivation for this lies in the aim of generating as many paths as possible during the MFPC-Kernousel initialization phase, resulting in a larger kernel set. Another reason is that, compared

with the problems addressed in the literature with CG, such as the Minimum Label Spanning Tree Problem [CGDC18], the Close-Enough Traveling Salesman Problem [CCCG20], the first choices performed by the GC algorithm in combinatorial optimization problems with conflict constraints are generally highly more impacting, since they lead to the exclusion of whole subsets of items from the ground set. As a result, a higher number of iterations is required. Clearly, the higher the number of conflicts, whose density is up to 60% in the addressed set of instances, the higher the impact of the first choices. On the other hand, the value of λ has been selected with a view of obtaining kernel sets able to give initial feasible solutions for all the instances of the problem, while the value of γ results from a trade-off between the number of variables in each bucket and the resulting number of subproblems to be solved.

			BDw			CG			KS			Kernousel		
id	**Istanza**	**Best**	**Value**	**Time**	**Gap**	**Value**	**Time**	**Gap**	**Value**	**Time**	**Gap**	**Value**	**Time**	**Gap**
1	40_30_30_10_15	37	37	11,93	0.00%	37	6.61	0.00%	37	10.7	0.00%	37	15.75	0.00%
2	40_30_30_15_20	49	49	3,73	0.00%	37	6.74	24.49%	49	6.7	0.00%	49	13.22	0.00%
3	40_30_40_10_15	24	24	8,02	0.00%	15	2.17	37.50%	24	10.61	0.00%	24	11.62	0.00%
4	40_30_40_15_20	33	33	4,60	0.00%	33	5.73	0.00%	33	10.29	0.00%	33	15.19	0.00%
5	40_30_50_10_15	12	12	6,26	0.00%	12	1.93	0.00%	12	12.98	0.00%	12	15.51	0.00%
6	40_30_50_15_20	34	34	5,56	0.00%	34	2.92	0.00%	34	9.92	0.00%	34	14.59	0.00%
7	40_30_60_10_15	13	13	13,84	0.00%	13	2.41	0.00%	13	13.58	0.00%	13	15.06	0.00%
8	40_30_60_15_20	15	15	19,39	0.00%	15	2.01	0.00%	15	15.52	0.00%	15	13.86	0.00%
9	40_40_30_10_15	35	35	13,86	0.00%	34	7.88	2.86%	34	14.1	2.86%	34	20.44	2.86%
10	40_40_30_15_20	51	51	165,45	0.00%	49	3.21	3.92%	51	26.97	0.00%	51	37.81	0.00%
11	40_40_40_10_15	26	26	18,93	0.00%	26	11.45	0.00%	26	16.54	0.00%	26	29.84	0.00%
12	40_40_40_15_20	50	50	16,33	0.00%	50	9.39	0.00%	50	17.32	0.00%	50	28.35	0.00%
13	40_40_50_10_15	13	13	28,22	0.00%	13	3.30	0.00%	13	19.89	0.00%	13	23.61	0.00%
14	40_40_50_15_20	19	19	20,61	0.00%	19	2.77	0.00%	19	19.01	0.00%	19	19.11	0.00%
15	40_40_60_10_15	25	25	24,05	0.00%	25	5.61	0.00%	25	16.49	0.00%	25	22.83	0.00%
16	40_40_60_15_20	19	19	27,18	0.00%	19	2.81	0.00%	19	21.17	0.00%	19	27.63	0.00%
17	40_50_30_10_15	43	43	132,96	0.00%	35	22.99	18.60%	36	73.83	16.28%	37	104.82	13.95%
18	40_50_30_15_20	55	55	23,58	0.00%	54	14.02	1.82%	52	17.24	5.45%	55	32.85	0.00%
19	40_50_40_10_15	33	33	26,96	0.00%	33	11.98	0.00%	33	24.75	0.00%	33	34.85	0.00%
20	40_50_40_15_20	35	35	37,47	0.00%	33	13.51	5.71%	33	18.15	5.71%	33	33.77	5.71%
21	40_50_50_10_15	27	27	50,04	0.00%	26	3.87	3.70%	27	32.86	0.00%	27	33.67	0.00%
22	40_50_50_15_20	34	34	34,87	0.00%	32	7.78	5.88%	34	26.56	0.00%	34	36.22	0.00%
23	40_50_60_10_15	15	15	23,51	0.00%	13	4.85	13.33%	15	12.83	0.00%	15	18	0.00%
24	40_50_60_15_20	34	34	24,46	0.00%	34	6.23	0.00%	34	13.18	0.00%	34	20.58	0.00%
25	40_60_30_10_15	45	45	479,46	0.00%	41	20.45	8.89%	45	23.39	0.00%	45	59.68	0.00%
26	40_60_30_15_20	80	80	72,87	0.00%	69	22.20	13.75%	80	25.86	0.00%	80	50.5	0.00%
27	40_60_40_10_15	30	30	120,64	0.00%	24	11.97	20.00%	30	36.03	0.00%	30	54.29	0.00%
28	40_60_40_15_20	48	48	120,80	0.00%	35	20.34	27.08%	38	23.12	20.83%	48	48.31	0.00%
29	40_60_50_10_15	26	26	52,23	0.00%	24	9.47	7.69%	26	26.31	0.00%	26	37.19	0.00%
30	40_60_50_15_20	36	36	59,77	0.00%	35	6.69	2.78%	34	34.96	5.56%	35	47.61	2.78%
31	40_60_60_10_15	24	24	17,98	0.00%	24	4.49	0.00%	24	25.24	0.00%	24	29.25	0.00%
32	40_60_60_15_20	37	37	6,74	0.00%	37	13.48	0.00%	37	12.68	0.00%	37	27.03	0.00%
33	50_30_30_10_15	37	37	16,53	0.00%	37	13.65	0.00%	37	17.64	0.00%	37	31.59	0.00%
34	50_30_30_15_20	48	48	31,91	0.00%	48	17.02	0.00%	47	17.06	2.08%	48	36.85	0.00%
35	50_30_40_10_15	24	24	29,55	0.00%	20	6.74	16.67%	24	16.75	0.00%	24	26.3	0.00%
36	50_30_40_15_20	33	33	43,10	0.00%	33	3.92	0.00%	33	14.26	0.00%	33	21.01	0.00%
37	50_30_50_10_15	21	21	30,47	0.00%	21	5.05	0.00%	21	18.27	0.00%	21	26.85	0.00%
38	50_30_50_15_20	33	33	28,92	0.00%	30	7.02	9.09%	33	21.78	0.00%	33	30.63	0.00%
39	50_30_60_10_15	13	13	18,75	0.00%	13	4.02	0.00%	13	8.39	0.00%	13	12.62	0.00%
40	50_30_60_15_20	37	37	9,62	0.00%	37	5.34	0.00%	37	7.76	0.00%	37	13.23	0.00%
41	50_40_30_10_15	37	37	570,34	0.00%	36	18.13	2.70%	31	31.4	16.22%	36	65.01	2.70%
42	50_40_30_15_20	53	53	115,33	0.00%	53	11.37	0.00%	38	30.19	28.30%	53	33.58	0.00%
43	50_40_40_10_15	35	35	185,13	0.00%	35	10.15	0.00%	35	32.21	0.00%	35	47	0.00%
44	50_40_40_15_20	38	38	53,69	0.00%	37	5.48	2.63%	37	26.65	2.63%	37	32.64	2.63%
45	50_40_50_10_15	21	21	72,35	0.00%	14	12.31	33.33%	21	34.19	0.00%	21	54.03	0.00%
46	50_40_50_15_20	18	18	44,01	0.00%	17	7.41	5.56%	17	19.16	5.56%	17	30.37	5.56%
47	50_40_60_10_15	24	24	14,72	0.00%	24	8.73	0.00%	24	21.55	0.00%	24	30.92	0.00%
48	50_40_60_15_20	37	37	24,22	0.00%	37	8.63	0.00%	37	24.61	0.00%	37	35.29	0.00%
49	50_50_30_10_15	40	40	352,72	0.00%	40	23.65	0.00%	39	44.72	2.50%	40	72.96	0.00%
50	50_50_30_15_20	57	57	127,67	0.00%	55	21.04	3.51%	55	30.21	3.51%	55	72.15	3.51%
51	50_50_40_10_15	40	40	163,82	0.00%	38	17.72	5.00%	40	30.31	0.00%	40	51.72	0.00%
52	50_50_40_15_20	53	53	134,36	0.00%	53	17.53	0.00%	53	28.61	0.00%	53	55.14	0.00%
53	50_50_50_10_15	25	25	64,95	0.00%	25	23.27	0.00%	25	31.87	0.00%	25	59.48	0.00%
54	50_50_50_15_20	38	38	61,36	0.00%	38	5.88	0.00%	38	32.89	0.00%	38	41.73	0.00%
55	50_50_60_10_15	25	25	38,60	0.00%	11	5.78	56.00%	25	21.07	0.00%	25	27.67	0.00%
56	50_50_60_15_20	18	18	17,43	0.00%	18	7.86	0.00%	18	20.75	0.00%	18	29.49	0.00%
57	50_60_30_10_15	49	45	2627,74	8.16%	48	65.62	2.04%	49	140.14	0.00%	49	412.85	0.00%
58	50_60_30_15_20	73	73	1986,50	0.00%	70	29.88	4.11%	73	60.9	0.00%	73	159.58	0.00%
59	50_60_40_10_15	38	38	294,45	0.00%	37	17.40	2.63%	34	82.64	10.53%	37	74.82	2.63%
60	50_60_40_15_20	51	51	679,74	0.00%	50	34.59	1.96%	49	62.49	3.92%	50	108.07	1.96%
61	50_60_50_10_15	39	39	123,31	0.00%	38	16.77	2.56%	38	40.02	2.56%	38	62.84	2.56%
62	50_60_50_15_20	37	37	183,23	0.00%	37	21.73	0.00%	37	47.4	0.00%	37	67.42	0.00%
63	50_60_60_10_15	26	26	67,36	0.00%	26	9.41	0.00%	26	29.99	0.00%	26	46.1	0.00%
64	50_60_60_15_20	33	33	58,24	0.00%	33	13.37	0.00%	33	37.96	0.00%	33	57.25	0.00%
65	60_30_30_10_15	36	36	850,40	0.00%	36	23.43	0.00%	34	30.31	5.56%	36	72.29	0.00%
66	60_30_30_15_20	53	53	418,12	0.00%	39	7.61	26.42%	50	19.88	5.66%	53	36.25	0.00%
67	60_30_40_10_15	22	22	56,08	0.00%	21	12.23	4.55%	22	22.21	0.00%	22	36.48	0.00%
68	60_30_40_15_20	34	34	22,39	0.00%	33	19.11	2.94%	34	16.24	0.00%	34	35.87	0.00%
69	60_30_50_10_15	27	27	16,67	0.00%	27	14.31	0.00%	27	23.63	0.00%	27	51.18	0.00%
70	60_30_50_15_20	33	33	31,44	0.00%	33	6.40	0.00%	33	25.41	0.00%	33	33.21	0.00%
71	60_30_60_10_15	12	12	19,94	0.00%	12	6.57	0.00%	12	20.78	0.00%	12	28.75	0.00%
72	60_30_60_15_20	19	19	21,45	0.00%	17	11.37	10.53%	18	22.51	5.26%	18	35.38	5.26%
73	60_40_30_10_15	39	39	2627,74	0.00%	36	27.16	7.69%	36	38.82	7.69%	36	91.87	7.69%
74	60_40_30_15_20	64	64	716,82	0.00%	64	27.17	0.00%	64	33.95	0.00%	64	67.04	0.00%
75	60_40_40_10_15	26	26	113,15	0.00%	26	19.47	0.00%	26	25.09	0.00%	26	47.22	0.00%
76	60_40_40_15_20	38	38	602,89	0.00%	37	15.33	2.63%	37	39.16	2.63%	37	59.99	2.63%
77	60_40_50_10_15	29	29	87,02	0.00%	28	15.39	3.45%	28	51.42	3.45%	29	70.69	0.00%
78	60_40_50_15_20	37	37	58,60	0.00%	37	17.55	0.00%	37	45.57	0.00%	37	64.69	0.00%
79	60_40_60_10_15	14	14	52,82	0.00%	12	8.57	14.29%	14	27.35	0.00%	14	42.95	0.00%
80	60_40_60_15_20	37	37	36,19	0.00%	34	7.95	8.11%	37	27.27	0.00%	37	39.37	0.00%
81	60_50_30_10_15	53	40	2627,74	24.53%	53	47.90	0.00%	41	55.92	22.64%	53	130.19	0.00%
82	60_50_30_15_20	68	52	2627,74	23.53%	64	49.82	5.88%	52	89.24	23.53%	64	200.59	5.88%
83	60_50_40_10_15	40	40	1322,13	0.00%	40	19.64	0.00%	36	71.58	10.00%	40	95.05	0.00%
84	60_50_40_15_20	39	39	550,96	0.00%	39	23.01	0.00%	39	41.43	0.00%	39	76.64	0.00%
85	60_50_50_10_15	26	26	494,15	0.00%	26	35.52	0.00%	26	56.48	0.00%	26	99.48	0.00%
86	60_50_50_15_20	37	37	197,58	0.00%	35	16.41	5.41%	37	51.69	0.00%	37	73.86	0.00%
87	60_50_60_10_15	25	25	70,09	0.00%	25	24.44	0.00%	25	43.68	0.00%	25	68.91	0.00%
88	60_50_60_15_20	19	19	129,96	0.00%	18	11.94	5.26%	19	43.24	0.00%	19	58.9	0.00%
89	60_60_30_10_15	54	53	2627,74	1.85%	54	46.95	0.00%	51	110.05	5.56%	54	238.59	0.00%
90	60_60_30_15_20	67	51	2627,74	23.88%	63	61.17	5.97%	51	161.29	23.88%	67	630.81	0.00%
91	60_60_40_10_15	41	41	1931,83	0.00%	41	65.82	0.00%	41	68.37	0.00%	41	166.78	0.00%
92	60_60_40_15_20	53	53	2100,13	0.00%	53	36.08	0.00%	53	66.86	0.00%	53	124.27	0.00%
93	60_60_50_10_15	27	27	719,36	0.00%	26	16.04	3.70%	27	65.94	0.00%	27	96.99	0.00%
94	60_60_50_15_20	53	53	406,50	0.00%	53	53.68	0.00%	53	69.61	0.00%	53	142.01	0.00%
95	60_60_60_10_15	24	24	177,19	0.00%	24	29.23	0.00%	24	59.86	0.00%	24	93.19	0.00%
96	60_60_60_15_20	33	33	201,56	0.00%	33	28.90	0.00%	33	63.14	0.00%	33	98.53	0.00%
AVG				**359.23**	**0.85%**		**16.19**	**4.71%**		**34.80**	**2.61%**		**63.84**	**0.71%**

Table 6.1: Small instances

ID	Instance	Best	BDw			CG			KS			Kernousel		
			Value	Time	Gap	Value	Time	Gap	Value	Time	Gap	Value	Time	Gap
97	70_30_30_10_15	39	39	917.60	0.00%	36	29.82	7.69%	35	33.61	10.26%	38	98.63	2.56%
98	70_30_30_15_20	35	35	2627.74	0.00%	35	18.11	0.00%	34	46.83	2.86%	35	92.15	0.00%
99	70_30_40_10_15	23	23	129.12	0.00%	22	8.68	4.35%	21	30.31	8.70%	22	46.44	4.35%
100	70_30_40_15_20	36	36	81.43	0.00%	36	39.74	0.00%	36	32.85	0.00%	36	75.31	0.00%
101	70_30_50_10_15	25	25	78.10	0.00%	12	8.48	52.00%	25	29.66	0.00%	25	41.21	0.00%
102	70_30_50_15_20	19	19	82.96	0.00%	16	8.16	15.79%	19	30.84	0.00%	19	41.96	0.00%
103	70_30_60_10_15	14	14	43.10	0.00%	14	7.06	0.00%	13	29.07	7.14%	14	38.46	0.00%
104	70_30_60_15_20	20	20	45.20	0.00%	20	15.43	0.00%	20	27.25	0.00%	20	44.96	0.00%
105	70_40_30_10_15	56	45	2627.74	19.64%	46	53.46	17.86%	45	49.31	19.64%	56	172.76	0.00%
106	70_40_30_15_20	65	52	2627.74	20.00%	65	42.01	0.00%	37	121.21	43.08%	65	129.78	0.00%
107	70_40_40_10_15	31	31	821.39	0.00%	31	25.27	0.00%	31	59.82	0.00%	31	92.34	0.00%
108	70_40_40_15_20	36	36	1451.58	0.00%	35	37.58	2.78%	35	41.49	2.78%	35	101.58	2.78%
109	70_40_50_10_15	26	26	359.03	0.00%	24	25.73	7.69%	26	60.72	0.00%	26	93.33	0.00%
110	70_40_50_15_20	33	33	427.15	0.00%	32	24.68	3.03%	20	61.08	39.39%	32	89.66	3.03%
111	70_40_60_10_15	25	25	155.78	0.00%	25	17.63	0.00%	25	50.29	0.00%	25	71.63	0.00%
112	70_40_60_15_20	18	18	107.77	0.00%	16	9.94	11.11%	18	49.69	0.00%	18	63.11	0.00%
113	70_50_30_10_15	41	34	2627.74	17.07%	40	56.68	2.44%	41	79	0.00%	40	373.96	2.44%
114	70_50_30_15_20	52	48	2627.74	7.69%	52	70.66	0.00%	49	61.86	5.77%	52	201.5	0.00%
115	70_50_40_10_15	35	35	2060.04	0.00%	35	75.53	0.00%	35	68.61	0.00%	35	178	0.00%
116	70_50_40_15_20	55	55	2627.74	0.00%	51	48.23	7.27%	54	82.24	1.82%	55	142.41	0.00%
117	70_50_50_10_15	25	25	1831.55	0.00%	21	26.02	16.00%	24	92.96	4.00%	24	122.67	4.00%
118	70_50_50_15_20	50	50	509.68	0.00%	50	14.12	0.00%	50	71.03	0.00%	50	94.79	0.00%
119	70_50_60_10_15	24	24	307.09	0.00%	23	64.10	4.17%	24	75.25	0.00%	24	151.91	0.00%
120	70_50_60_15_20	19	19	593.09	0.00%	18	13.78	5.26%	19	74.09	0.00%	19	93.91	0.00%
121	70_60_30_10_15	54	41	2627.74	24.07%	54	95.74	0.00%	41	238.65	24.07%	54	566.2	0.00%
122	70_60_30_15_20	86	55	2627.74	36.05%	71	92.36	17.44%	58	202.65	32.56%	86	345.7	0.00%
123	70_60_40_10_15	37	34	2627.74	8.11%	36	55.00	2.70%	37	106.03	0.00%	37	201.89	0.00%
124	70_60_40_15_20	54	52	2627.74	3.70%	52	40.21	3.70%	53	93.19	1.85%	52	169.81	3.70%
125	70_60_50_10_15	32	32	1739.77	0.00%	32	59.19	0.00%	28	101.93	12.50%	32	198.19	0.00%
126	70_60_50_15_20	53	53	2403.17	0.00%	51	65.83	3.77%	46	106.13	13.21%	51	201.68	3.77%
127	70_60_60_10_15	25	25	426.82	0.00%	25	79.62	0.00%	25	110.7	0.00%	25	211.27	0.00%
128	70_60_60_15_20	36	36	483.96	0.00%	36	59.57	0.00%	36	108.55	0.00%	36	197.31	0.00%
129	80_30_30_10_15	38	31	2627.74	18.42%	34	51.05	10.53%	32	77.46	15.79%	34	168.43	10.53%
130	80_30_30_15_20	69	69	2081.23	0.00%	68	28.74	1.45%	56	46.08	18.84%	69	101.84	0.00%
131	80_30_40_10_15	23	23	1449.07	0.00%	22	14.85	4.35%	22	48.63	4.35%	22	75.39	4.35%
132	80_30_40_15_20	37	37	545.77	0.00%	34	39.75	8.11%	33	51.51	10.81%	37	111.38	0.00%
133	80_30_50_10_15	15	15	245.84	0.00%	12	11.01	20.00%	14	45.64	6.67%	14	66.85	6.67%
134	80_30_50_15_20	50	50	143.39	0.00%	50	36.66	0.00%	36	51.61	28.00%	50	100.92	0.00%
135	80_30_60_10_15	15	15	90.98	0.00%	15	26.19	0.00%	15	46.84	0.00%	15	85.53	0.00%
136	80_30_60_15_20	34	34	115.27	0.00%	32	21.07	5.88%	18	54.95	47.06%	32	83.21	5.88%
137	80_40_30_10_15	54	54	2627.74	0.00%	54	59.86	0.00%	54	66.38	0.00%	54	157.24	0.00%
138	80_40_30_15_20	65	49	2627.74	24.62%	65	67.17	0.00%	52	76.05	20.00%	65	248.38	0.00%
139	80_40_40_10_15	29	29	2627.74	0.00%	27	58.31	6.90%	29	88.38	0.00%	29	188.85	0.00%
140	80_40_40_15_20	50	50	2627.74	0.00%	45	47.17	10.00%	38	77	24.00%	50	148.68	0.00%
141	80_40_50_10_15	28	28	724.61	0.00%	23	27.12	17.86%	26	75.84	7.14%	26	114.31	7.14%
142	80_40_50_15_20	50	50	1078.21	0.00%	50	50.27	0.00%	36	87.38	28.00%	50	148.08	0.00%
143	80_40_60_10_15	26	26	267.57	0.00%	25	27.67	3.85%	26	83.15	0.00%	26	117.93	0.00%
144	80_40_60_15_20	34	34	387.61	0.00%	32	17.39	5.88%	34	80.13	0.00%	34	106.12	0.00%
145	80_50_30_10_15	56	40	2627.74	28.57%	48	86.57	14.29%	49	101.88	12.50%	56	252.13	0.00%
146	80_50_30_15_20	64	48	2627.74	25.00%	61	137.46	4.69%	50	107.5	21.88%	64	601.95	0.00%
147	80_50_40_10_15	39	38	2627.74	2.56%	38	74.36	2.56%	28	120.29	28.21%	38	209.93	2.56%
148	80_50_40_15_20	55	38	2627.74	30.91%	51	87.08	7.27%	38	110.92	30.91%	51	235.12	7.27%
149	80_50_50_10_15	25	25	2627.74	0.00%	24	41.02	4.00%	25	119.96	0.00%	25	176.54	0.00%
150	80_50_50_15_20	35	35	2482.31	0.00%	35	24.75	0.00%	35	117.68	0.00%	35	149.68	0.00%
151	80_50_60_10_15	23	23	1040.82	0.00%	23	36.08	0.00%	14	130.86	39.13%	23	175.44	0.00%
152	80_50_60_15_20	33	33	1087.28	0.00%	19	24.16	42.42%	19	125.86	42.42%	19	158.79	42.42%
153	80_60_30_10_15	60	41	2627.74	31.67%	57	155.41	5.00%	47	153.39	21.67%	60	1376.66	0.00%
154	80_60_30_15_20	88	68	2627.74	22.73%	85	100.28	3.41%	88	134.35	0.00%	85	501.96	3.41%
155	80_60_40_10_15	38	33	2627.74	13.16%	33	117.96	13.16%	38	155.13	0.00%	38	303.17	0.00%
156	80_60_40_15_20	53	37	2627.74	30.19%	46	93.48	13.21%	53	154.84	0.00%	53	286.24	0.00%
157	80_60_50_10_15	38	38	2627.74	0.00%	38	67.94	0.00%	38	173.85	0.00%	38	253.1	0.00%
158	80_60_50_15_20	48	37	2627.74	22.92%	37	45.70	22.92%	37	181	22.92%	37	241.46	22.92%
159	80_60_60_10_15	26	26	1824.90	0.00%	26	47.24	0.00%	25	183.08	3.85%	26	245.92	0.00%
160	80_60_60_15_20	36	36	1395.80	0.00%	35	58.09	2.78%	36	191.75	0.00%	36	254.18	0.00%
AVG				**1536.52**	**6.05%**		**47.97**	**6.49%**		**89.79**	**10.37%**		**190.47**	**2.18%**

Table 6.2: Large instances

Computational results are shown in Tables 6.1 and 6.2, reporting information about the tests performed on the Small and Large instances, respectively. The first three columns of Tables 6.1 and 6.2 report the instance ID, the instance name and the value of the best known feasible solution (Best), respectively. Then, for each of the four compared approach, namely Benders Decomposition (BD_w), Carousel Greedy (GC), Kernel Search (KS) and MFPC-Kernousel, three additional columns are reported, indicating: the value associated with the solution identified by that approach (*Value*), the execution time in seconds (*Time*), and the percentage gap measuring how far, in terms of value, the identified solution is from the best known one (*Gap*). The best known solution is the one, among the solutions identified by any of the compared approaches, yielding the higher objective function value.

On Small instances, the average percentage gap of the solutions identified by the CG, with respect to the best known solutions, is 4.71%, the KS approach achieves a gap of 2.61%, while the Kernousel approach achieves an average gap of 0.71%. On Large instances, the average percentage gap achieved by CG, KS and Kernousel are 6.49%, 10.37% and 2.18%, respectively. The obtained results show the increase of the solution quality resulting from the combination of CG and KS. On the other side, the state-of-the-art BD_w method reports an average percentage gap, with respect to the best known solution, of 0.85% on the Small instances and 6.05% on Large instances. The average percentage gap values produced by the Kernousel approach are lower than the ones yielded by BD_w on both classes of instances, even though such difference is more evident on Large instances. Furthermore, concerning the computational times, the Kernousel requires, on average, 63.84 seconds to solve a small instance and 190.47 to solve a large one. Thus, the proposed approach is faster than the state-of-the-art method, which requires, on average, 359.23 and 1536.52 seconds to solve a small and large instance, respectively.

A cumulative chart, is shown in Figure 6.4, reports percentage gap values, on the x-axis, and the number of instances solved within a given gap value w.r.t the best known feasible solution, on the y-axis. When the BD_w approach does not identify an optimal solution, it generally performs worse than Kernousel. Indeed, the number of instances for which the produced solution has an associated gap lower or equal to ρ, with $\rho \in [3\%, 24\%]$, is higher for the Kernousel approach compared with all the other approaches. Although the BD_w method globally identifies more optima, it generally yields a higher gap value than Kernousel when it does not achieve an optimal solution.

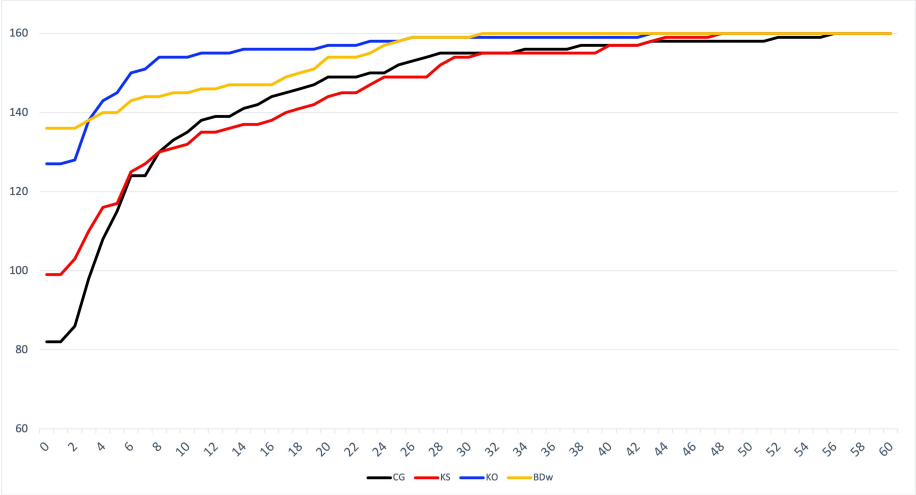


Figure 6.4: Cumulative chart of percentage gap with respect to the best known solution.

Chapter 7

Minimum Spanning Tree with Conflicts

The Minimum Spanning Tree Problem (MST) consists in finding the spanning tree of an edge-weighted graph, with the minimum possible total edge weight. In this chapter, we deal with an NP-Hard variant of MST, named Minimum Spanning Tree Problem with Conflicts (MSTC), where, given a list of conflicting edges, the goal is to find the cheapest spanning tree with no edges in conflict.

MSTC has several practical applications. For instance, this problem arises in the context of network design of offshore wind farm where the overlap of cables should be avoided. This last condition can be modeled with the conflict contraints among the edges. Another application concerns is the analysis of the road maps with forbidden transitions [KLM13], i.e., maps where some routes are not valid because, for example, there are positions where it is forbidden to turn left or right. Another example concerns the installation of an oil pipeline system connecting various countries where conflicts arise because there are several firms and countries involved in the process. [DPST09].

In [DPST09, DPSW11] the authors faced the MSTC by using a conflict graph to represent the conflict edge pairs. In a conflict graph, each node represents an edge of the original graph and two nodes are connected if their corresponding edges, in the original graph, are in conflict. The authors prove that MSTC is strongly NP-hard, but still polynomially solvable if every connected component of the conflict graph is a single edge (i.e., a path of length one). Other special cases of MSTC solvable in polynomial time are studied in [ZKP11]. Regarding the heuristic approaches for MSTC, a GRASP method, coupled with adaptive memory programming, is presented in [BPOR19], and a multiethnic genetic algorithm is proposed in [CCP19].In [SU15], a general preprocessing method and a branch-and-cut algorithm are proposed with the generation of cutting planes corresponding to subtour elimination constraints and odd-cycle inequalities. Another branch-and-cut method for the MSTC is proposed in [CCPR21a], where a new set of valid inequalities for the problem is introduced. To obtain upper bounds that help pruning the search tree, the authors of [CCPR21a] use the genetic algorithm proposed in [CCP19]. As in [ZKP11], in [CG21b], the Lagrangian relaxation of the MSTC is studied and solved with an ad hoc dual ascent procedure.

In this chapter we introduce a new solution approach for the MSTC based on the Kernel Search method [AMS12]. Moreover we introduce a new variant of the problem named Minimum Spanning Tree Problem with Conflicts Placement (MSTCP). In this new variant, we assume that the decision on the pairs in conflict is made by a different entity w.r.t. the one deciding on the minimum spanning tree, and we model this scenario using Mixed-Integer Bilevel Linear Programming (MIBLP).

To the best of our knowledge, it is the first time that bilevel optimization is used to model this particular interaction, even if it has been used to deal with MST in different contexts. In most of the cases, as in our work, the follower is constructing the MST after the decisions of the leader. What differs in the various problems is the role of the leader. In our approach, the leader defines the conflicting edges. In [FSO99] two versions of the MST interdiction problem are presented. In the first one, which is NP-hard, the leader wants to find the set of k edges, which removal maximizes the MST weight. In the second one, which is polynomially solvable, the leader wants to find the finite increase in the edges weights that, again, maximizes the MST weight. In [Gas02, Gas09], the leader can either decrease or increase the edge weights, and its objective function consists of the weight modification costs. A pricing setting is studied in [CDF+11], where the leader maximizes its revenue by setting prices for a set of edges.

In [SZP19, BHH22] a completely different bilevel problem is considered: the leader and the follower choose a spanning tree together, according to different objective functions. This problem finds applications in the transportation network design problem, where some connections should be constructed by the central government, and some others should be constructed by a local government. The two agents have clearly different construction costs and objectives. In [BHH22] this problem is proved to be NP-hard.

In [SPR22], single-level mixed integer linear programming reformulations for some variants of bilevel MST are derived.

7.1 Integer Linear Programming Formulation of MSTC

Let us consider an undirected graph $\mathcal{G} = (V, E)$, where $V = \{1, \dots, n\}$ is the set of nodes and $E = \{e_1, \dots, e_m\}$ the set of edges. The edge e_i can be defined also by its two endpoints $u, v \in V$. For a given subset of nodes $S \subseteq V$, $E(S) = \{\{u, v\} \in E \mid u, v \in S\}$ is the set of edges having both the two endpoints in S. Let C be the set of conflicting edge pairs, which we call *conflict set*:

$$C = \{\{e_i, e_j\} \ : \ e_i, e_j \in E, \text{ and } e_i \text{ is in conflict with } e_j\}.$$

The general formulation for MSTC uses the following binary variables, defined for each i s.t. $e_i \in E$:

$$x_i = \begin{cases} 1 & \text{if edge } e_i \text{ is included in the tree} \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

The ILP formulation of the MSTC, coming from the *Subtour Elimination* formulation of the classical MST, reads:

$$\min_x \sum_{i:e_i \in E} w_i x_i \tag{7.2a}$$

$$\text{s.t.} \sum_{i:e_i \in E} x_i = n-1 \tag{7.2b}$$

$$\sum_{i:e_i \in E(S)} x_i \leq |S|-1 \qquad \forall\, S \subset V, |S| \geq 3 \tag{7.2c}$$

$$x_i + x_j \leq 1 \qquad \forall\, i,j : \{e_i, e_j\} \in C \tag{7.2d}$$

$$x \in \{0,1\}^{|E|} \tag{7.2e}$$

The objective function (7.2a) is representing the cost of the spanning tree, which we want to minimize. Constraint (7.2b) imposes that $n-1$ edges should be selected, since a spanning tree should include exactly $|V|-1$ edges. The exponentially many constraints (7.2c) are needed to ensure that no subtour is contained in the selected edges, i.e., no subgraph induced by S contains a cycle. Finally, constraints (7.2d) impose that, for each pair of edges in conflict, at most one edge of such pair can be selected.

In the following, we denote by LP-MSTC the Linear Programming (LP) relaxation of model (7.2), obtained by relaxing constraints (7.2e) to $x \in [o,1]^{|E|}$. Given a subset of arcs $F \subseteq E$, we further denote by MSTC(F) the ILP restriction of probem (7.2) obtained by restricting the set of variables to the arcs in F, i.e., adding constraints

$$x_i = 0 \quad \forall i \in E \setminus F$$

to formulation (7.2).

A polynomial alternative way to prevent subtours is obtained introducing additional (continuous) variables f_{ij} representing the amount of "flow" moved on arc (i,j) of the directed graph $\mathcal{G}' = (V, A)$ derived from $\mathcal{G}$ as follows. Starting from the set of edges E, the set of directed arcs A is defined by considering, for each edge in E with endpoints u and v, the two corresponding directed arcs (u,v), and (v,u). Furthermore, the set $\delta^+(u) \subseteq A$ is the set of all the arcs exiting from node u, while the set $\delta^-(u) \subseteq A$ is the set of all the arcs entering in node u.

In order to prevent subtours, one should impose that a flow of one unit is sent from the root node (arbitrarily set to node 1) to every other node, with the following constraints:

$$\sum_{(1,v)\in\delta^+(0)} f_{1v} - \sum_{(v,1)\in\delta^-(1)} f_{v1} = n-1 \tag{7.3a}$$

$$\sum_{(u,v)\in\delta^+(u)} f_{uv} - \sum_{(v,u)\in\delta^-(u)} f_{vu} = -1 \quad \forall i = 2,\dots,n \tag{7.3b}$$

$$0 \leq f_{uv} \leq (n-1)x_i \quad \forall (u,v) \in A : e_i = \{u,v\} \in E \tag{7.3c}$$

7.2 Kernel Search approach

In this section, we describe the Kernel Search (KS) framework we use to solve MSTC problem. Given the mathematical formulation (7.2) of MSTC, the starting point of KS algorithm consists in identifying a subset of promising variables of this model, i.e., variables that have a high probability of being non-zero in the optimal solution of the problem. These variables are used to form a subset called *kernel*, while the remaining ones are partitioned into several subsets called *buckets*. We define Λ^0 the set of indices of the variables in the initial kernel, with $K = |\Lambda^0|$, and B_k the set of indices of the variables in the k-th bucket for $k = 1, \dots, b$ ($\mathcal{B}$ is the initial set of all the buckets), with $S = |B_k|$ for all k, and $b = |\mathcal{B}|$.

The partitioning of the variables indices into the kernel and the buckets is carried out by KS according to the optimal solution of LP-MSTC. After the definition of these sets, KS seeks for an initial feasible solution by solving the MSTC restricted to the variables having indices inside the kernel only, i.e., solving MSTC(Λ^0). For this reason, the initial kernel size K should be, at the same time, large enough to contain as many variables as possible which are likely to be part of an optimal solution, and not too large to ensure that MSTC(Λ^0) can be efficiently solved to optimality.

The kernel is then updated throughout the KS algorithm (Λ_k is the kernel at the end of iteration k), by taking into account the buckets. At each iteration, the MILP formulation restricted to the current kernel and a bucket from the sequence is solved, trying to obtain solutions of increasing quality.

The KS algorithm is divided into two phases:

- **Initialization phase**: The initial set Λ^0 and the bucket sequence $\mathcal{B} = \{B_1, \dots B_b\}$ are built. Then, the MSTC(Λ^0) is solved and, if a feasible solution is found, its objective function value is set as the current upper bound w^* of the optimal solution we are looking for.

- **Extension phase**: At each iteration k, new variables from buckets are added to the kernel iterating (eventually more than once) over the buckets, and the MSTC restricted on the so-obtained enlarged kernel $\tilde{\Lambda}^k$ is solved. If MSTC($\tilde{\Lambda}^k$) is feasible, if the solution value is lower than w^* then w^* is updated with this new value.

Let us describe in more detail, how these two phases work. The construction of Λ^0 and $\mathcal{B}$ is carried out by creating a sorted sequence of the LP-MSTC variables. In particular, the variables which are basic in the optimal solution of LP-MSTC are sorted according to their values, in decreasing order, and placed at the beginning of the sequence. Instead, the variables which are non-basic in the solution of LP-MSTC, are sorted according to their reduced cost coefficient, in decreasing order, and are added at the end of the sequence.

Once the parameters K and S are set, Λ^0 is populated with the first K variables indices of the previously sorted sequence, and the remaining variables are equally distributed to populate the buckets $B_1, \dots, B_b$.

The initialization phase is concluded by solving the MSTC restricted to the variables whose indices are in Λ^0. If a feasible solution is found, its value is set as the upper bound w^* of the optimal solution we are looking for.

The optimal value of which is stored as the current upper bound on the MSTC value w^*. With the initial kernel Λ^0 and the initial set of buckets $\mathcal{B}$, the extension phase starts. During this phase, both the kernel and the set of buckets are updated. The initial kernel Λ^0 is enlarged using the buckets: at each iteration k, an *enlarged* bucket $\tilde{B}_k$ is populated in Algorithm 21, and added to the kernel; the MSTC restricted to the current kernel is solved. At each iteration of the kernel search the size of the buckets changes; once the buckets have been generated, in the first iteration (iteration 1), the algorithm uses one bucket at a time, in the second iteration it uses two buckets at a time (iteration 2), proceeding in this way until it reaches the maximum number of iterations (in the our implementation equal to 3). If an optimal solution is found then the subset of variables of the current bucket $\tilde{B}_k$, which are nonzero in this optimal solution, is added to the kernel and removed from the original buckets in Algorithm 22, i.e., the indices belonging to subset $\hat{B}_k$ are added to Λ^{k-1}, and removed from B_i with $i \in 1, \ldots, b$.

In this way, the algorithm tries to improve the quality of the solution through a refined search that involves, at each iteration k, the solution of MSTC restricted to the variables in $\tilde{\Lambda}_k$, given by the union of $\Lambda^{k-1} = \Lambda^0 \cup \hat{B}_1 \cup \cdots \cup \hat{B}_{k-1}$, and the k-th enlarged bucket $\tilde{B}_k$. This is done by adding to MSTC(Λ^{k-1}) the following constraint:

$$\sum_{i \in \tilde{B}_k} x_i \geq 1. \tag{7.4}$$

Once the new MSTC restriction is solved, if its optimal value w^k is better than the current upper bound w^*, we update such upper bound setting $w^* = w^k$.

The iteration over the buckets is performed at most three times. After the first iteration is concluded (i.e., all the buckets B_k in $\mathcal{B}$ are taken into account and updated by excluding the basic variables in $\hat{B}_k$), another iteration is done by considering two buckets at a time. Finally, if no feasible solution has been found yet, a third iteration is performed, by considering three buckets at a time.

The KS method stops if three iterations over the buckets have been performed or if a fixed number of consecutive iterations are carried out without improving the current best upper bound.

We report a pseudocode of our KS method in Algorithm 19 (Initialization Phase) and Algorithm 20.

Algorithm 19: KS algorithm: Initialization Phase

Data: Given $\mathcal{G} = (V, E), K, S, \Lambda^0 = \emptyset, B_i = \emptyset$ for $i = 1, \ldots, b$, $L = \emptyset$.
Result: Initial kernel Λ^0, initial buckets set $\mathcal{B}$. First solution found x^0 and its optimal value w^0.
1 Compute the optimal solution $\bar{x}$ of LP-MSTC model. Compute sets $G = \{i \in E : \bar{x}_i > 0\}$ and $N = \{i \in E : \bar{x}_i = 0\}$.
2 Sort indices $i \in G$ according to the values of $\bar{x}_i$
3 Sort indices $i \in N$ according to the values of the reduced cost coefficients of $\bar{x}_i$
4 $L = (G, N)$
5 $\Lambda^0 = L[0 : K]$
6 Populate buckets B_i by taking S variables at a time from $L[K + 1 : |E|]$. Being b the number of nonempty buckets, let $\mathcal{B} = \{B_1, \ldots, B_b\}$.
7 **if** *MSTC*(Λ^0) *is feasible* **then**
8 Compute the optimal solution x^0 of MSTC(Λ^0), and its optimal value w^0.
9 **else**
10 $x^0 = 0$, and $w^0 = +\infty$
11 **end**
12 **return** $\Lambda^0, \mathcal{B}, x^0, w^0$.

Algorithm 19 takes in input a graph $\mathcal{G} = (V, E)$, two integer values K and S representing the size of kernel and buckets respectively, various empty subsets of E, i.e., Λ^0, B_i for $i \in b$, and finally the empty vector L, which will be used to store the ordered variables indices. In line 1 we solve the LP relaxation of MSTC, i.e., LP-MSTC, obtaining the sets G and N of basic and non-basic variables indices, respectively. In lines 2 and 3, we order the variables indices w.r.t. the criterion previously described: the G indices are sorted in decreasing order of corresponding variables values, the N indices in decreasing order of corresponding variables reduced cost coefficients. In line 4, L is obtained by concatenating the two ordered sets of indices. In line 5, Λ^0 is populated with the variables indices contained in L from position zero to K. The remaining variables indices in L are used in line 6 to populate the buckets, taking S indices for each bucket. The set $\mathcal{B}$ is defined as the set of buckets. In line 7, the initialization phase is concluded solving the MSTC(Λ^0) formulation, if feasible. The algorithm returns its optimal value w^0 and its optimal solution x^0, as well as sets Λ^0 and $\mathcal{B}$.

Then, the extension phase in Algorithm 20 starts. It requires in input the initial solution x^0 (which is the best known solution x^*) and value w^0 (which is the best known lower bound w^*), the initial kernel Λ_0, the buckets set $\mathcal{B}$, and the δ value which is used as termination criterion. The outer-iteration index j (representing the iteration over the buckets) and the number of iterations in which no progress is made $\bar{k}$ are set to one and zero respectively. The algorithm returns the best found solution x^* and its objective function value w^*.

Algorithm 20: KS algorithm: Extension Phase

Data: Given $x^0, w^0, \Lambda^0, \mathcal{B} = \{B_1, \ldots, B_b\}, \delta > 0$, let $x^* = x^0, w^* = w^0$, $j = 1$, $\bar{k} = 0$, J.
Result: Optimal solution found x^* and its optimal value w^*.

1 **while** $j \leq 3$ **do**
2 $k = 1$
3 **while** $k \leq b$ **do**
4 Define the current bucket $\tilde{B}_k$ Algorithm 21 Build the k-th kernel $\tilde{\Lambda}^k = \Lambda^{k-1} \cup \tilde{B}_k$
5 **if** *MSTC*($\tilde{\Lambda}^k$) *is feasible* **then**
6 Compute the optimal solution x^k of MSTC($\tilde{\Lambda}^k$), and its optimal value w^k.
7 **else**
8 $w^k = +\infty$
9 **end**
10 **if** $w^k < w^*$ **then**
11 $x^* = x^k$, and $w^* = w^k$
12 Set $\hat{B}_k = \{i \in \tilde{B}_k : x^k_i > 0\}$
13 $\Lambda^k = \Lambda^{k-j} \cup \hat{B}_k$
14 Update the original buckets in $\mathcal{B}$ with Algorithm 22.
15 $\bar{k} = 0$
16 **else**
17 $\bar{k} = \bar{k} + 1$
18 **end**
19 $k = k + j$
20 **if** $\bar{k} \geq \delta$ *and* $w^* < +\infty$ **then**
21 $k = b + 1$
22 $j = J + 1$
23 **end**
24 **end**
25 $j = j + 1$
26 $\Lambda^0 = \Lambda^k$
27 **end**
28 **return** w^*, x^*

Index j is used to indicate the outer-iterations over the buckets. If $j = 1$, one bucket is considered at a time. If $j = 2$, two consecutive buckets are considered at a time. Finally, if $j = 3$, three consecutive buckets are taken into account at a time. At each inner-iteration k, the set Λ^{k-1} is enlarged using bucket $\tilde{B}_k$ (defined by Algorithm 21), obtaining the set $\tilde{\Lambda}^k$, and the MSTC($\tilde{\Lambda}^k$) formulation, if feasible, is solved (line 6). If its optimal value w^k is s.t. $w^k < w^*$; we set $x^* = x^k$ and $w^* = w^k$; we define Λ^k as $\Lambda^{k-1} \cup \hat{B}_k$, where $\hat{B}_k$ is the set of indices of the variables in $\tilde{B}_k$ which are nonzero in x^k; finally, we remove from set B_k such subset $\hat{B}_k$ in Algorithm 22, being variables in such set already added to the kernel, and set $\bar{k} = 0$. If $w^k \geq w^*$ instead, we increase parameter $\bar{k}$, which is needed for the stopping criterion of the algorithm.

If $\bar{k} > \delta$, (lines 20–23) i.e., for more than δ steps, the solution has not been improved, and at least a feasible solution has been found (the upper bound w^* is not infinite) the algorithm terminates.

We emphasize that, by solving the LP-MSTC, we have a lower bound on the optimal value of MSTC, and at each iteration of the kernel algorithm, if the current restriction MSTC($\tilde{\Lambda}^k$) is feasible, we have a valid upper bound for the problem, with the associated feasible solution.

Algorithm 21: Population of the enlarged bucket $\tilde{B}_k$.

Data: Given $\mathcal{B} = \{B_1, \ldots, B_b\}$, j, k, and the set $\hat{B}^k \subseteq \tilde{B}^k$ set $\tilde{B}_k = \emptyset$, $l = 0$.
Result: Enlarged bucket $\tilde{B}_k$.

```
while l < j do
    if k + l ≤ b then
        B̃_k = B̃_k ∪ B_{k+l}
        l = l + 1
    else
        break
    end
end
l = 0
return B̃_k
```

Algorithm 22: Update of the original buckets in $\mathcal{B}$.

Data: Given $\mathcal{B} = \{B_1, \ldots, B_b\}$, j, k, set $l = 0$.
Result: Updated set $\mathcal{B}$.

```
while l < j do
    if k + l ≤ b then
        B_{k+l} = B_{k+l} \ B̂_k
        l = l + 1
    else
        break
    end
end
l = 0
return B̃_k
```

7.2.1 Valid inequalities and separation procedures

In this section, we present some valid inequalities for the MSTC taken from [CCPR21a] we add to our model. These inequalities may help in the solution of the subproblems solved throughout the kernel algorithm. Indeed, at each iteration, a restriction of the MSTC is solved through branch-and-cut type of algorithms, which make use of valid inequalities or cuts introduced at the nodes of a branch-and-bound tree to progressively tighten the formulation.

The first type of inequalities, defined *degree-cut inequalities*, explicitly impose that, for each node of the graph, at least one incident edge should be selected in the solution:

$$\sum_{i:e_i \in N(u)} x_i \geq 1 \quad \forall u \in V \tag{7.5}$$

These constraints are not really useful when dealing with integer solutions. However, if solving a linear relaxation of the original formulation, these may help in finding a better bound of the optimal value. These $|V|$ inequalities are added to the MSTC model (7.2) as a priori constraints.

The *conflict-cycle inequalities* are a stronger version of the subtour elimination constraints (7.2c) obtained by taking into account also the conflicts among the edges. Indeed, given a cycle $P \subset E$, constraints (7.2c) impose that the edges belonging to P selected in any feasible solution of the problem should be at most $|P|-1$ (not all of them can be selected, otherwise a subtour is included in the solution). If an edge e_c outside the cycle P is in conflict with two edges belonging to the cycle P, then in any feasible solution of MSTC, the number of selected edges in $P \cup \{e_c\}$ should be still less than $|P| - 1$: no cycle, as well as no pairs of edges in conflict should be included in the solution. This is imposed by the following inequalities:

$$\sum_{i:e_i \in P} x_i + x_c \leq |P| - 1 \quad \forall \text{ cycle } P \subset E, e_c \in E \setminus P. \tag{7.6}$$

These inequalities, as well as the classical subtour elimination constraints (7.2c), are exponentially many, thus a separation procedure should be implemented for them. In particular, the separation problem for constraints (7.2c), which consists in finding the violated constraints of type (7.2c) by a given solution $\bar{x}$, corresponds to solving a maximum-flow (minimum-cut) problem in a correspondingly defined capacitated graph as described in [PW83]. As regards conflict-cycle inequalities (7.6), the heuristic separation procedure proposed in [CCPR21a] is implemented.

Finally, we consider the so-called *odd-cycle inequalities*, which are obtained by considering the conflict graph $\mathcal{G}'$, as the graph having nodes corresponding to the edges of the original graph, and edges representing the conflicts, i.e., two nodes i and j in the conflict graph are connected by an edge iff they correspond to two conflicting edges e_i, e_j in the original graph. If a solution of the MSTC includes two connected vertices in the conflict graph, it violates one of the constraints (7.2d). Thus, the following inequalities holds:

$$\sum_{i:e_i \in P'} x_i \leq \frac{|P'| - 1}{2} \quad \forall P' \subset E \text{ inducing an odd cycle in } \mathcal{G}' \tag{7.7}$$

where $P' \subseteq E$ corresponds to the nodes of the conflict graph $\mathcal{G}'$ inducing an odd cycle. These inequalities are separated by using the exact algorithm proposed in [GS86].

7.3 Bilevel Programming formulation of MSTCP

There may be a situation in which the set of conflicting edges is not given. Instead, there is an agent who can decide the conflicts among edges with the aim of maximizing the resulting minimimum spanning tree.

We can model this problem using bilevel programming. A bilevel program can be seen as a hierarchical game with two players, a leader and a follower, making their decisions in a hierarchical order. In our case, the leader wants to maximize the weight of the minimum

spanning tree obtained by putting exactly B pairs of edges in conflict. The follower detects the the cheapest spanning tree with no edges in conflict. The follower's variables are the x_i variables defined in Eq. (7.1), while the leader's binary variables are

$$y_{ij} = \begin{cases} 1 & \text{if the pair } (e_i, e_j) \text{ is in conflict} \\ 0 & \text{otherwise.} \end{cases}$$

The lower-level feasible set, without considering the linking constraint modeling the conflicting pairs, is:

$$\left\{ x_i \in \{0,1\}^{|E|} : \sum_{i:e_i \in E} x_i = n-1, \right. \\ \left. \sum_{i:e_i \in S} x_i \leq |S| - 1 \quad \forall S \subseteq V, |S| \leq 3 \right\}$$

which corresponds to the feasible set of the classical subtour elimination formulation of the MST.

Instead, using the single-commodity flow formulation of the MST, the lower-level feasible set reads:

$$\begin{aligned} \mathcal{X} = \Bigg\{ x_i \in \{0,1\}^{|E|}, f_{uv} \in \mathbb{R}^{2|E|} : & \sum_{i:e_i \in E} x_i = |V| - 1, \\ & \sum_{(0,v) \in \delta^+(0)} f_{0v} - \sum_{(v,0) \in \delta^-(0)} f_{v0} = n-1, \\ & \sum_{(u,v) \in \delta^+(u)} f_{uv} - \sum_{(v,u) \in \delta^-(u)} f_{vu} = -1 \quad \forall i = 1, \ldots, n, \\ & 0 \leq f_{uv} \leq (n-1)x_i \qquad \forall e_i = \{u,v\} \in E, \\ & 0 \leq f_{vu} \leq (n-1)x_i \qquad \forall e_i = \{u,v\} \in E \Bigg\} \end{aligned}$$

The upper-level feasible set is:

$$\mathcal{Y} = \left\{ y \in \{0,1\}^{|E| \times |E|} : \sum_{i=1}^{|E|} \sum_{j=1}^{|E|} y_{ij} = B \right\}$$

modeling the fact that the leader must select exactly B conflicts. Since the leader and the follower share the same objective function, the bilevel formulation of the MSTC problem reads

$$\max_{y \in \mathcal{Y}} \min_{x,f \in \mathcal{X}} \left\{ \sum_{i:e_i \in E} w_i x_i : x_i + x_j \leq 2 - y_{ij} \quad \forall (e_i, e_j) \in E \times E \right\}. \tag{7.8}$$

If, in formulation (7.8), we set $B = |C|$ and add at the upper level the following constraints : $y_{ij} = 1 \; \forall i, j \; : \; (e_i, e_j) \in C$, and $y_{ij} = 0 \; \forall i, j \; : \; (e_i, e_j) \in E \setminus C$, i.e., giving to variable y a specific value, solving the corresponding bilevel formulation corresponds to solving formulation (7.2).

In order to deal with MIBLP problems, ad-hoc methods for the specific case may be constructed, either aiming to reformulate it into a single-level problem or to solve it mainly via heuristics, branch-and-bound, or branch-and-cut approaches. In our work, we decide to use the general purpose solver proposed in [FLMS17], which exploits a branch-and-cut framework to solve Mixed-Integer Bilevel Linear problems.

7.4 Computational Tests

7.4.1 MSTC

In this section, we report the results obtained by solving all the instances proposed in [CCPR21a]with the Kernel Search Method Algorithms (19–20). The instances introduced in [CCPR21a] have number of nodes in $\{25, 50, 75, 100\}$ and number of edges s.t. the density of the graph is equal to 0.2, 0.3, or 0.4. The number of conflicts pairs $|C|$ is equal to 1%, 4%, 7% of $m(m-1)/2$ (graph density). The parameter s is the seed used to initialize the random number generator. For each combination of these parameters (scenario) the authors generated 5 instances by varying the seed used for the generation of random numbers, generating in total 180 instances.

We compare our results to the ones obtained on the same instances by using the heuristic proposed in [CG21b]. This heuristic provides a lower bound on the optimal value of the problem by solving the Lagrangian dual of the MSTC, and an upper bound through a local search procedure. Furthermore, we report the value of the optimal solution (for the instances solved to optimality) or the value of the best known feasible solution (for the instances not solved to optimality) found by the branch-and-cut methods proposed in [CCPR21a].

In these tests we set:

$$K = \max \left\{ |V| - 1, \left\lfloor \frac{|E|}{2} \right\rceil \right\} \quad S = \left\lfloor 0.15 * |E| \right\rceil.$$

These parameters have been identified through a tuning phase. The selected value of K ensures that the starting kernel contains a sufficient number of variables to provide a feasible solution of the problem plus possible alternatives. The size S of the buckets is chosen in such a way that the subproblems solved at each iteration are not excessively large. Furthermore, we set the parameter δ, indicating the maximum number of iterations the KS can perform without improving, to $\lfloor |\mathcal{B}| \times 0.2 \rfloor$. Each subproblem involved in the KS algorithm is solved using the state-of-the-art solver Cplex.

The KS is coded in Python on an OSX platform, 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 (Version identifier: 22.1.0.0) solver (single thread mode). We set a time limit of three hours.

Table 7.1 shows the computational results of the KS on the instances proposed in [CCPR21a], compared, again, to the results obtained by the branch-and-cut method proposed in [CCPR21a], but also by the Lagrangian based method proposed in [CG21b].

The headings of Table 7.1 are the following: n is the number of nodes of the graph; m is the number of edges of the graph; $|C|$ is the number of conflicts; b is the number of created buckets; B&C is the optimal solution value or the best solution (with the "$*$" symbol) found by the Branch&Cut algorithm proposed in [CCPR21a]; LM is the value of the solution found by the Lagrangian based method introduced in [CG21b]; w^* is the value of the best feasible solution found by the KS algorithm 19–20; time[s] is the computational time (in seconds) required by the KS; gap_B&C[%] is the percentage gap with respect to the value found by the Branch&Cut, i.e., gap_B&C $= 100\frac{w^* - \text{B\&C}}{w^*}$; gap_LM[%] is the percentage gap with respect to the value found by the Lagrangian based Methods, i.e., gap_LM $= 100\frac{w^* - \text{LM}}{w^*}$;

As shown in Table 7.1, the KS approach finds, on average, better results than the Lagrangian based method, with an average gap_LM[%] of -11.60%. Indeed, KS finds 55 optima out of the 107 found by the B&C, against the 45 found by the Lagrangian. For 4 instances (identified by the red color in Table 7.1) the KS found a better feasible solution value than the one returned by the B&C. However the KS does not find all the optima found by this exact approach. On the instances in which the B&C finds an optimal solution, and KS does not, the average gap_B&C is 0.54%, i.e., the KS gets very close to the optimum. If we consider instead also the instances in which the B&C finds at least a feasible solution, this average gap value increases up to 12.36%, i.e., on the instances in which also the B&C struggles in finding the optimal solution, the KS does not perform better.

Table 7.1: Comparison of the results on the instances proposed in [CCPR21a]

Instance				KS Setting	Literature		KS		Comparison	
ID	n	m	$\|C\|$	b	B&C	LM	w^*	time[s]	gap_B&C[%]	gap_LM[%]
51	25	60	18	4	347	347	347	0.0	0.00	0.00
52	25	60	18	4	389	389	389	0.0	0.00	0.00
53	25	60	18	4	353	353	353	0.4	0.00	0.00
54	25	60	18	4	346	346	346	0.0	0.00	0.00
55	25	60	18	4	336	336	336	0.0	0.00	0.00
56	25	60	71	4	381	381	381	0.4	0.00	0.00
57	25	60	71	4	390	390	393	0.4	0.76	0.76
58	25	60	71	4	372	372	372	0.0	0.00	0.00
59	25	60	71	4	357	357	357	0.1	0.00	0.00
60	25	60	71	4	406	406	406	0.0	0.00	0.00
61	25	60	124	4	385	385	385	0.0	0.00	0.00
62	25	60	124	4	432	432	432	0.0	0.00	0.00
63	25	60	124	4	458	474	458	0.4	0.00	-3.49
64	25	60	124	4	400	400	405	0.4	1.23	1.23
65	25	60	124	4	420	421	420	0.3	0.00	-0.24
66	25	90	41	4	311	311	311	0.5	0.00	0.00
67	25	90	41	4	306	306	306	0.0	0.00	0.00
68	25	90	41	4	299	299	299	0.4	0.00	0.00
69	25	90	41	4	297	297	297	0.5	0.00	0.00
70	25	90	41	4	318	318	320	0.6	0.63	0.63
71	25	90	161	4	305	305	305	0.5	0.00	0.00
72	25	90	161	4	339	339	339	0.0	0.00	0.00
73	25	90	161	4	344	344	344	0.1	0.00	0.00
74	25	90	161	4	329	331	330	0.4	0.30	-0.30
75	25	90	161	4	326	327	330	1.1	1.21	0.91
76	25	90	281	4	349	349	350	0.5	0.29	0.29
77	25	90	281	4	385	385	385	0.8	0.00	0.00
78	25	90	281	4	335	335	335	0.6	0.00	0.00
79	25	90	281	4	348	358	348	0.9	0.00	-2.87
80	25	90	281	4	357	359	356	0.9	-0.28	-0.84
81	25	120	72	4	282	282	282	0.0	0.00	0.00
82	25	120	72	4	294	294	294	0.3	0.00	0.00
83	25	120	72	4	284	284	284	0.0	0.00	0.00
84	25	120	72	4	281	281	281	0.5	0.00	0.00

Table 7.1: Comparison of the results on the instances proposed in [CCPR21a]

Instance				KS Setting	Literature		KS		Comparison	
ID	n	m	$\|C\|$	b	B&C	LM	w^*	time[s]	gap_B&C[%]	gap_LM[%]
85	25	120	72	4	292	292	292	0.0	0.00	0.00
86	25	120	286	4	321	321	322	0.4	0.31	0.31
87	25	120	286	4	317	317	317	1.2	0.00	0.00
88	25	120	286	4	284	284	284	0.0	0.00	0.00
89	25	120	286	4	311	312	311	0.3	0.00	-0.32
90	25	120	286	4	290	290	290	0.3	0.00	0.00
91	25	120	500	4	329	341	331	1.4	0.60	-3.02
92	25	120	500	4	339	347	342	1.8	0.88	-1.46
93	25	120	500	4	368	383	368	1.4	0.00	-4.08
94	25	120	500	4	311	314	310	0.8	-0.32	-1.29
95	25	120	500	4	321	325	324	1.5	0.93	-0.31
96	50	245	299	5	619	619	619	6.6	0.00	0.00
97	50	245	299	5	604	604	604	4.8	0.00	0.00
98	50	245	299	5	634	634	634	0.0	0.00	0.00
99	50	245	299	5	616	616	616	6.4	0.00	0.00
100	50	245	299	5	595	595	595	4.0	0.00	0.00
101	50	245	1196	5	678	698	687	6.3	1.31	-1.60
102	50	245	1196	5	681	721	702	13.5	2.99	-2.71
103	50	245	1196	5	709	725	718	6.3	1.25	-0.97
104	50	245	1196	5	639	656	651	14.6	1.84	-0.77
105	50	245	1196	5	681	748	689	9.5	1.16	-8.56
106	50	245	2093	5	833*	-	821	280.5	-1.46	-100
107	50	245	2093	5	835	-	872	372.0	4.24	-100
108	50	245	2093	5	840*	-	824	381.2	-1.94	-100
109	50	245	2093	5	836*	-	854	311.4	2.11	-100
110	50	245	2093	5	769	-	809	23.8	4.94	-100
111	50	367	672	5	570	570	570	7.0	0.00	0.00
112	50	367	672	5	561	561	561	17.5	0.00	0.00
113	50	367	672	5	573	573	573	11.6	0.00	0.00
114	50	367	672	5	560	560	560	8.1	0.00	0.00
115	50	367	672	5	549	551	549	11.3	0.00	-0.36
116	50	367	2687	5	612	657	621	36.2	1.45	-5.80
117	50	367	2687	5	615	663	638	8.9	3.61	-3.92
118	50	367	2687	5	587	635	591	7.4	0.68	-7.45
119	50	367	2687	5	634	721	655	12.6	3.21	-10.08
120	50	367	2687	5	643	688	649	18.5	0.92	-6.01
121	50	367	4702	5	726*	-	792	1236	8.33	-100
122	50	367	4702	5	770*	-	813	1068	5.29	-100
123	50	367	4702	5	786*	-	821	1847	4.26	-100
124	50	367	4702	5	711*	-	762	1860	6.69	-100
125	50	367	4702	5	764*	868	827	1856	7.62	-4.96
126	50	490	1199	6	548	552	548	24.2	0.00	-0.73
127	50	490	1199	6	530	531	530	24.8	0.00	-0.19
128	50	490	1199	6	549	549	550	32.5	0.18	0.18
129	50	490	1199	6	540	541	549	0.0	1.64	1.46
130	50	490	1199	6	540	540	540	24.5	0.00	0.00
131	50	490	4793	6	594	629	603	43.8	1.49	-4.31
132	50	490	4793	6	579	650	582	49.4	0.52	-11.68
133	50	490	4793	6	589	657	590	27.7	0.17	-11.36
134	50	490	4793	6	577	643	590	40.9	2.20	-8.98
135	50	490	4793	6	592	670	607	37.0	2.47	-10.38
136	50	490	8387	6	678*	812	800	702.9	15.25	-1.50
137	50	490	8387	6	651*	-	737	770.2	11.67	-100
138	50	490	8387	6	689*	-	734	1726	6.13	-100
139	50	490	8387	6	682*	-	801	943.2	14.86	-100
140	50	490	8387	6	674*	828	728	257.0	90.80	-13.74
141	75	555	1538	5	868	869	871	47.1	0.34	0.23
142	75	555	1538	5	871	878	878	106.6	0.80	0.00
143	75	555	1538	5	838	844	839	7.3	0.12	-0.60
144	75	555	1538	5	855	855	858	124.6	0.35	0.35
145	75	555	1538	5	857	859	858	56.7	0.12	-0.12
146	75	555	6150	5	1047*	1236	1161	2908	9.82	-6.46
147	75	555	6150	5	1069*	1207	1129	2604	5.31	-6.91
148	75	555	6150	5	1040*	-	1103	1598	5.71	-100
149	75	555	6150	5	998*	-	1096	3027	8.94	-100
150	75	555	6150	5	994*	-	1038	2260	4.24	-100
151	75	555	10762	5	-	-	-	2407	-	-
152	75	555	10762	5	-	-	-	2623	-	-
153	75	555	10762	5	-	-	-	2270	-	-
154	75	555	10762	5	-	-	-	2738	-	-
155	75	555	10762	5	-	-	-	2077	-	-
156	75	832	3457	6	798	803	800	271.6	0.25	-0.38
157	75	832	3457	6	821	832	823	89.1	0.24	-1.09
158	75	832	3457	6	816	820	816	75.7	0.00	-0.49
159	75	832	3457	6	820	822	823	1251	0.36	0.12

Table 7.1: Comparison of the results on the instances proposed in [CCPR21a]

Instance				KS Setting	Literature		KS		Comparison	
ID	n	m	$\|C\|$	b	B&C	LM	w^*	time[s]	gap_B&C[%]	gap_LM[%]
160	75	832	3457	6	815	827	817	104.6	0.24	-1.22
161	75	832	13828	6	903*	1131	1029	1521.9	12.24	-9.91
162	75	832	13828	6	953*	1125	1072	830.5	11.10	-4.94
163	75	832	13828	6	892*	1035	989	1635	9.81	-4.65
164	75	832	13828	6	915*	1140	1002	1503	8.68	-13.77
165	75	832	13828	6	896*	1057	1106	1624	18.99	4.43
166	75	832	24199	6	-	-	-	4873	-	-
167	75	832	24199	6	-	-	-	4563	-	-
168	75	832	24199	6	-	-	-	5048	-	-
169	75	832	24199	6	-	-	-	4988	-	-
170	75	832	24199	6	-	-	-	5404	-	-
171	75	1110	6155	6	787	788	787	173.0	0.00	-0.13
172	75	1110	6155	6	785	786	787	1319	0.25	0.13
173	75	1110	6155	6	783	800	784	823.6	0.13	-2.04
174	75	1110	6155	6	784	789	784	336.7	0.00	-0.64
175	75	1110	6155	6	797	809	806	120.3	1.12	-0.37
176	75	1110	24620	6	867*	1036	897	1361	3.34	-15.50
177	75	1110	24620	6	851*	1048	902	3252	5.65	-16.19
178	75	1110	24620	6	892*	1098	922	1282	3.25	-19.09
179	75	1110	24620	6	864*	1076	1017	2116	15.04	-5.80
180	75	1110	24620	6	882*	1073	995	2139	11.36	-7.84
181	75	1110	43085	6	-	-	-	7620	-	-
182	75	1110	43085	6	-	-	-	7259	-	-
183	75	1110	43085	6	1194*	-	-	7744	100.00	-
184	75	1110	43085	6	-	-	-	7733	-	-
185	75	1110	43085	6	-	-	-	7530	-	-
186	100	990	4896	6	1119	1163	1122	229.3	0.27	-3.65
187	100	990	4896	6	1137	1156	1143	239.3	0.52	-1.14
188	100	990	4896	6	1113	1143	1120	314.9	0.63	-2.05
189	100	990	4896	6	1110	1155	1112	921.7	0.18	-3.87
190	100	990	4896	6	1090	1114	1098	772.9	0.73	-1.46
191	100	990	19583	6	-	-	-	7322	-	-
192	100	990	19583	6	1491*	-	-	6892	100	-
193	100	990	19583	6	1510*	-	-	7214	100	-
194	100	990	19583	6	1441*	-	-	7258	100	-
195	100	990	19583	6	1560*	-	-	7291	100	-
196	100	990	34269	6	-	-	-	3920	-	-
197	100	990	34269	6	-	-	-	4158	-	-
198	100	990	34269	6	-	-	-	3960	-	-
199	100	990	34269	6	-	-	-	3881	-	-
200	100	990	34269	6	-	-	-	4148	-	-
201	100	1485	11019	6	1079	1136	1081	775.5	0.19	-5.09
202	100	1485	11019	6	1056	1084	1058	619.1	0.19	-2.46
203	100	1485	11019	6	1059	1077	1059	951.9	0.00	-1.70
204	100	1485	11019	6	1046	1059	1047	1021	0.10	-1.15
205	100	1485	11019	6	1072	1109	1074	1556	0.19	-3.26
206	100	1485	44075	6	1374*	-	-	8077	100	-
207	100	1485	44075	6	1291*	-	-	7935	100	-
208	100	1485	44075	6	1344*	-	-	8178	100	-
209	100	1485	44075	6	1286*	-	-	8075	100	-
210	100	1485	44075	6	1370*	-	-	7957	100	-
211	100	1485	77131	6	-	-	-	8948	-	-
212	100	1485	77131	6	-	-	-	8122	-	-
213	100	1485	77131	6	-	-	-	8240	-	-
214	100	1485	77131	6	-	-	-	7455	-	-
215	100	1485	77131	6	-	-	-	8375	-	-
216	100	1980	19593	6	1031	1069	1031	1890	0.00	-3.69
217	100	1980	19593	6	1036	1081	1037	2227	0.10	-4.24
218	100	1980	19593	6	1024	1064	1024	1573	0.00	-3.91
219	100	1980	19593	6	1025	1043	1025	1949	0.00	-1.76
220	100	1980	19593	6	1028	1076	1028	1413	0.00	-4.67
221	100	1980	78369	6	1234*	-	-	9577	100	-
222	100	1980	78369	6	1187*	1680	-	10758	100	100
223	100	1980	78369	6	1213*	-	-	9361	100	-
224	100	1980	78369	6	1221*	-	-	10402	100	-
225	100	1980	78369	6	1245*	1686	-	9898	100	100
226	100	1980	137145	6	-	-	-	10600	-	-
227	100	1980	137145	6	-	-	-	10600	-	-
228	100	1980	137145	6	-	-	-	10600	-	-
229	100	1980	137145	6	-	-	-	10600	-	-
230	100	1980	137145	6	-	-	-	10600	-	-

7.4.2 MSTCP

In order to solve the bilevel formulation (7.8) of MSTCP, we use the general purpose solver for bilevel problems proposed in [FLMS17]. It is a branch-and-cut approach for solving bilevel problems, which discard bilevel infeasible solutions by adding intersection cuts on the fly. Since the follower's variables are both binary and continuous (f variables are continuous) the *hypercube-type* intersection cuts are used.

All experiments in this section are conducted in multi-threaded mode, on a 2.3GHz Intel Xeon E5 CPU, 128GB RAM, allowing the solver to use 35 out of 40 cores.

We use the bilevel solver to solve formulation (7.8) with $B = |C|$ (i.e. using the same number of conflicts used for the CMST problem version), using the instances proposed in [CCPR21a], already described in Section 7.4.1, only with $n \in \{25, 50\}$, as this version of the problem is combinatorially more difficult than the CMST. A time limit of three hours of computation and no memory limit is enforced for each run. The results show that on average the solver fails to improve the upper bound after two hours of computation.

The computational results are reported in the tables 7.2 and 7.3, which contain the following columns: n, i.e., number of nodes; m, i.e., number of arches; B, i.e., the budget; UB, i.e., upperbound; LB, i.e., lower bound; *root_ bound*, i.e., the root lower bound; *time[s]*, i.e., computing time in seconds ; *root_ time[s]*, i.e., the root time in seconds; *root_ gap[%]*, i.e., the gap of the branch-and-cut root node and *final_ gap[%]*, i.e., final gap obtained from the bilevel solver.

Table 7.2 contains the computational results performed on the instances with $n = 25$, on which the bilevel solver identifies only 5 optima, of which 4 are identified on the root node. In 16 out of 45 cases, the optimum is identified in the root node. In the remaining 40 instances it always identifies a feasible solution obtaining an average final gap of 72.81%. Table 7.3 contains the computational results performed on the instances with $n = 50$. in this case the bilevel solver is able to identify only a few optima solutions 2 out of 45 of which only one is identified in the root node, obtaining an average final gap of 136,43%.

n	m	B	UB	LB	root_bound	time[s]	root_time[s]	root_gap[%]	final_gap[%]
25	60	18	590	590	590	129,73	0,13	0,00%	0,00%
25	60	18	423	626	627,67	10800,31	0,13	48,38%	47,99%
25	60	18	408	606	607,36	10800,3	0,14	48,86%	48,53%
25	60	18	407	622	622	10800,34	0,16	52,83%	52,83%
25	60	18	360	600	600	16342,49	0,64	66,67%	66,67%
25	60	71	395	582	582,67	10800,29	0,48	47,51%	47,34%
25	60	71	409	604	607,92	10800,3	0,14	48,63%	47,68%
25	60	71	497	623,74	624,7	10800,21	0,22	25,69%	25,50%
25	60	71	609	609	609	671,67	0,66	0,00%	0,00%
25	60	71	444	611	611	10801,49	0,31	37,61%	37,61%
25	60	124	564	564	564	5879,08	0,39	0,00%	0,00%
25	60	124	621	621	622	2980,14	0,24	0,16%	0,00%
25	60	124	440	629,62	631,98	10800,17	0,16	43,63%	43,10%
25	60	124	631	632	635	10801,86	0,14	0,63%	0,16%
25	60	124	577	610	610	10800,72	0,19	5,72%	5,72%
25	90	41	308	625	625,75	10800,86	0,24	103,17%	102,92%
25	90	41	335	672	672	10801,33	0,22	100,60%	100,60%
25	90	41	302	649	649	10800,93	0,28	114,90%	114,90%
25	90	41	303	638	638	10801,06	0,35	110,56%	110,56%
25	90	41	618	647,93	647,93	10800,51	0,27	4,84%	4,84%
25	90	161	318	650	650,59	10801,09	0,23	104,59%	104,40%
25	90	161	357	655,17	655,91	10800,46	0,24	83,73%	83,52%
25	90	161	340	669	669	10801,01	0,29	96,76%	96,76%
25	90	161	324	645	646	10800,65	0,24	99,38%	99,07%
25	90	161	339	646,71	648,65	10800,52	0,31	91,34%	90,77%
25	90	281	357	638	641	10801,07	0,32	79,55%	78,71%
25	90	281	376	678	678	10801,06	0,26	80,32%	80,32%
25	90	281	337	644	644	10800,8	0,21	91,10%	91,10%
25	90	281	371	644	644	10801,13	0,27	73,58%	73,58%
25	90	281	444	675	675	10800,9	0,28	52,03%	52,03%
25	120	72	660	663	664	10802,09	0,37	0,61%	0,45%
25	120	72	297	661,5	663	10801,55	0,37	123,23%	122,73%
25	120	72	285	683,17	684	10802,03	0,32	140,00%	139,71%
25	120	72	287	680	680	10802,3	0,41	136,93%	136,93%
25	120	72	299	655	656,78	10802,43	0,38	119,66%	119,06%
25	120	286	330	677	677	10802,42	0,37	105,15%	105,15%
25	120	286	306	672	672	10801,99	0,38	119,61%	119,61%
25	120	286	291	676	676	10801,95	1,78	132,30%	132,30%
25	120	286	311	653,94	654	10802,3	0,4	110,29%	110,27%
25	120	286	314	681	681	10802,81	0,38	116,88%	116,88%
25	120	500	317	677	677	10801,93	0,39	113,56%	113,56%
25	120	500	676	676	676	10802,31	1,09	0,00%	0,00%
25	120	500	334	681	682,75	10802,17	0,4	104,42%	103,89%
25	120	500	291	670	670	10801,9	0,37	130,24%	130,24%
25	120	500	302	659,83	659,91	10802,71	0,38	118,51%	118,49%
Average			402,31	643,86	644,52	10179,01	0,35	72,98%	72,81%
#OPT								4	5

Table 7.2: MSTCP problem tests conducted with $B = |C|$, on instances with $n = 25$.

n	m	B	UB	LB	root_bound	time[s]	root_time[s]	root_gap[%]	final_gap[%]
50	245	299	610	1377	1377	10818,95	2,26	125,74%	125,74%
50	245	299	594	1383,27	1385,55	10819,9	1,34	133,26%	132,87%
50	245	299	623	1407,92	1408	10816,36	1,7	126,00%	125,99%
50	245	299	609	1353	1354,02	10815,87	1,57	122,34%	122,17%
50	245	299	585	1362	1362,42	10816,01	1,63	132,89%	132,82%
50	245	1196	610	1365	1365	10818,39	1,58	123,77%	123,77%
50	245	1196	597	1375,86	1376	10827,52	8,74	130,49%	130,46%
50	245	1196	612	1361	1361	10819,25	2,02	122,39%	122,39%
50	245	1196	579	1351,86	1351,93	10820,81	2,69	133,49%	133,48%
50	245	1196	594	1337,94	1338,94	10816,58	1,77	125,41%	125,24%
50	245	2093	590	1370,91	1370,95	10820,33	5,85	132,36%	132,36%
50	245	2093	617	1374,88	1375	10818,5	1,83	122,85%	122,83%
50	245	2093	588	1346	1346	10817,23	1,69	128,91%	128,91%
50	245	2093	1362	1366	1366	10814,25	1,5	0,29%	0,29%
50	245	2093	1309	1381	1381	10819,23	1,44	5,50%	5,50%
50	367	672	562	1410,61	1413,92	10932,86	3,56	151,59%	151,00%
50	367	672	549	1429	1429,76	10934,25	4,08	160,43%	160,29%
50	367	672	556	1402	1402	10960,16	3,22	152,16%	152,16%
50	367	672	553	1418	1418	10913,22	2,78	156,42%	156,42%
50	367	672	543	1382	1382	10934,01	4,28	154,51%	154,51%
50	367	2687	552	1408,95	1408,95	10957,29	4,11	155,24%	155,24%
50	367	2687	546	1422,88	1424	10914,44	2,81	160,81%	160,60%
50	367	2687	530	1420	1420,5	10907,04	3,03	168,02%	167,92%
50	367	2687	553	1412,78	1412,78	10923,37	4,57	155,47%	155,47%
50	367	2687	589	1406,77	1407	10952,14	3,4	138,88%	138,84%
50	367	4702	570	1415,91	1416,16	10937,16	3,59	148,45%	148,41%
50	367	4702	1423	1423	1423,97	326,84	3,31	0,07%	0,00%
50	367	4702	576	1425,99	1426,49	10914,69	4,42	147,65%	147,57%
50	367	4702	549	1405	1405	10939	5,11	155,92%	155,92%
50	367	4702	602	1415	1415	10933,22	2,94	135,05%	135,05%
50	490	1199	539	1422,88	1423	11276,07	6,5	164,01%	163,98%
50	490	1199	525	1435	1435	11240,8	4,59	173,33%	173,33%
50	490	1199	543	1443,73	1443,73	11255,6	5,1	165,88%	165,88%
50	490	1199	533	1443	1443	11372,87	7,99	170,73%	170,73%
50	490	1199	537	1433	1433	11196,03	5,38	166,85%	166,85%
50	490	4793	546	1424,97	1425	11232,55	7,6	160,99%	160,98%
50	490	4793	531	1429	1429	11285,79	5,58	169,11%	169,11%
50	490	4793	541	1440,01	1440,08	11277,16	7,14	166,19%	166,18%
50	490	4793	531	1438	1438	11352,76	6,62	170,81%	170,81%
50	490	4793	529	1430	1430	11240,56	6,31	170,32%	170,32%
50	490	8387	540	1435	1435	11150,13	7,43	165,74%	165,74%
50	490	8387	531	1437	1437	11264,18	12,31	170,62%	170,62%
50	490	8387	1438	1438	1438	4167,85	5,05	0,00%	0,00%
50	490	8387	550	1442	1442	11302,28	5,43	162,18%	162,18%
50	490	8387	551	1425	1425,5	11404,53	6,53	158,71%	158,62%
Average			637,71	1405,07	1405,37	10615,07	4,28	136,49%	136,43%
#OPT								1	2

Table 7.3: MSTCP problem tests conducted with $B = |C|$, on instances with $n = 50$.

In order to further verify the quality of the solution obtained in the bilevel setting, we set the budget B to a value such that the leader can impose to the follower to return the worst possible MST, i.e., the maximum spanning tree. This value is given by $(n-1)(m-n+1) + \binom{m-n+1}{2}$, where the first term represents the number of edge pairs (e_i, e_j) s.t. e_i belongs to a spanning tree and e_j does not, and the second term represent the number of unique pairs in the set of edges not in the spanning tree. If the budget is set to this value, the leader will thus be able to put the $(m-n+1)$ edges outside the maximum spanning tree in conflict with all the remaining edges in the graph, forcing the follower to construct the MST equal to the maximum spanning tree. Given the results recorded in the previous tests (setting $B = |C|$) the tests were performed with a time limit of two hours and no memory limit.

The computational results are reported in the tables 7.4 and 7.5, which contain the same columns as the tables 7.2 and 7.3 plus one more column *obj_ maxMST*, i.e., the objective function value of the maximum MST.

Table 7.4 contains the computational results performed on the instances with $n = 25$, on which the bilevel solver always identifies the optimum (out of 41 instances) except in 4 cases where it fails to provide a feasible solution. In 16 out of 45 cases, the optimum is identified in the root node. Table 7.5 contains the computational results performed on the instances with $n = 50$. In this case the bilevel solver is able to identify only a few optimal solutions 8 out of 45 of which 5 are identified in the root node, in all other cases it is unable to identify a feasible solution.

Compared with the tests performed with a value of $B = |C|$, the bilevel solver has identified a smaller number of feasible solutions in these last tests, as increasing the budget B, the number of feasible solutions decreases, at the same time it has identified a greater number of optima.

In future developments, we want to implement an ad-hoc bilevel solution approach for the MSTCP problem, in order to overcome some weaknesses of the general purpose approaches, which proves to be a valid tool that allows the fast prototyping of bilevel models managing to obtain good results on instances of small and medium size.

n	m	B	UB	LB	root_bound	time[s]	root_time[s]	root_gap[%]	final_gap[%]	obj_maxMST
25	60	1494	590	590	591	0,29	0,19	1,7%	0%	590
25	60	1494	626	626	627	1,42	0,17	1,6%	0%	626
25	60	1494	606	606	606,99	2,9	0,16	1,6%	0%	606
25	60	1494	622	622	622	0,53	0,17	0%	0%	622
25	60	1494	600	600	600	0,17	0,17	0%	0%	600
25	60	1494	582	582	582,67	0,75	0,21	1,1%	0%	582
25	60	1494	604	604	607,92	580,16	0,18	6,5%	0%	604
25	60	1494		622,11	624,7	7200,06	0,15			614
25	60	1494	609	609	609	0,41	0,15	0%	0%	609
25	60	1494	611	611	611,5	1,14	0,18	0,8%	0%	611
25	60	1494	564	564	567,67	1,19	0,2	6,5%	0%	564
25	60	1494	621	621	621,95	1,22	0,17	1,5%	0%	621
25	60	1494	628	628	631,6	1509,8	0,16	5,7%	0%	628
25	60	1494	632	632	634,74	4,68	0,16	4,3%	0%	632
25	60	1494	610	610	610	0,21	0,14	0%	0%	610
25	90	3729	625	625	627,75	1,05	0,32	4,4%	0%	625
25	90	3729	672	672	672	0,74	0,28	0%	0%	672
25	90	3729	649	649	649	0,47	0,23	0%	0%	649
25	90	3729	638	638	638,5	0,92	0,29	0,8%	0%	638
25	90	3729		648	649,33	7200,07	0,36			646
25	90	3729	650	650	650,5	2,21	0,29	0,8%	0%	650
25	90	3729	654	654	655,91	6016,47	0,26	2,9%	0%	654
25	90	3729	669	669	669,5	0,88	0,3	0,7%	0%	669
25	90	3729		646	648,42	7200,06	0,27			645
25	90	3729		646,52	648,79	7200,06	0,34			640
25	90	3729	638	638	642,46	871,7	0,34	7,0%	0%	638
25	90	3729	678	678	678	1,87	0,32	0%	0%	678
25	90	3729	644	644	644	0,48	0,23	0%	0%	644
25	90	3729	644	644	644	0,94	0,26	0%	0%	644
25	90	3729	675	675	675,5	1,06	0,24	0,7%	0%	675
25	120	6864	663	663	664	5,47	0,5	1,5%	0%	663
25	120	6864	661	661	664	4059,6	0,48	4,5%	0%	661
25	120	6864	683	683	684	2300,57	0,65	1,5%	0%	683
25	120	6864	680	680	680,09	3,49	0,52	0,1%	0%	680
25	120	6864	655	655	656	7,59	0,44	1,5%	0%	655
25	120	6864	677	677	677	1,06	1,05	0%	0%	677
25	120	6864	672	672	672	1,12	1,12	0%	0%	672
25	120	6864	676	676	676	1,63	0,39	0%	0%	676
25	120	6864	653	653	655	216,52	0,47	3,1%	0%	653
25	120	6864	681	681	681	1,84	0,53	0%	0%	681
25	120	6864	676	676	677	4,6	0,38	1,5%	0%	676
25	120	6864	676	676	676	1,18	1,17	0%	0%	676
25	120	6864	681	681	681	1,22	1,21	0%	0%	681
25	120	6864	670	670	670	3,05	0,52	0%	0%	670
25	120	6864	659	659	659,91	2008,48	0,47	1,4%	0%	659
Average			644,00	643,70	644,79	1031,59	0,37	0,32%	0%	643,31
#OPT								16	41	

Table 7.4: Tests carried out in order to obtain the maximum MST with $n = 25$.

n	m	B	UB	LB	root_bound	time[s]	root_time[s]	root_gap[%]	final_gap[%]	obj_maxMST
50	245	28714	1377	1377	1377	21,26	4,08	0%	0%	590
50	245	28714		1384,36	1385,55	7200,21	3,25			626
50	245	28714		1408	1408	7200,23	5,63			606
50	245	28714		1353,26	1354,02	7200,2	2,22			622
50	245	28714		1362	1363	7200,2	2,96			600
50	245	28714		1365	1365	7200,22	4,04			582
50	245	28714		1376	1376	7200,19	3,42			604
50	245	28714		1361,97	1361,97	7200,21	4,04			614
50	245	28714		1351,93	1352	7200,27	3,66			609
50	245	28714		1339	1339,5	7200,24	3,62			611
50	245	28714		1369,88	1370,42	7200,18	3,41			564
50	245	28714	1374	1374	1374,94	38,01	2,86	0,7%	0%	621
50	245	28714		1346,05	1346,83	7200,18	3,81			628
50	245	28714	1366	1366	1366,5	8,63	3,08	0,4%	0%	632
50	245	28714		1380,71	1381	7200,2	3,83			610
50	367	65985		1413,94	1413,99	7200,43	9,26			625
50	367	65985		1429,95	1430,7	7200,42	7,94			672
50	367	65985		1402	1402	7200,39	8,01			649
50	367	65985	1418	1418	1418	76,1	5,22	0%	0%	638
50	367	65985		1384	1384,5	7200,45	5,22			646
50	367	65985		1409	1409	7200,39	7,67			650
50	367	65985		1423,87	1424	7200,38	5,45			654
50	367	65985	1420	1420	1420,5	66,3	8,31	0,4%	0%	669
50	367	65985		1412	1412,78	7200,39	9,18			645
50	367	65985		1407	1407	7200,44	8,74			640
50	367	65985		1415,74	1416,15	7200,39	9,06			638
50	367	65985		1424,5	1424,5	7200,4	8,66			678
50	367	65985		1427	1427,5	7200,4	8,56			644
50	367	65985		1405	1405	7200,48	11,58			644
50	367	65985	1415	1415	1415	81,12	8,92	0%	0%	675
50	490	118629		1423	1423	7200,63	16,94			663
50	490	118629		1435,94	1435,94	7200,69	11,52			661
50	490	118629		1443,73	1443,73	7200,59	16,41			683
50	490	118629		1442,98	1443	7200,71	22,68			680
50	490	118629		1433	1433	7200,68	13,84			655
50	490	118629		1424	1425,5	7200,65	16,8			677
50	490	118629	1429	1429	1429	348,43	29,47	0%	0%	672
50	490	118629		1440,08	1440,08	7200,67	14,88			676
50	490	118629	1438	1438	1438	219,8	15,71	0%	0%	653
50	490	118629		1430,84	1430,84	7200,71	16,37			681
50	490	118629		1435	1435	7200,62	28,98			676
50	490	118629		1437	1437	7200,61	25,11			676
50	490	118629		1439	1439	7200,7	12,8			681
50	490	118629		1442	1442,5	7200,66	20,32			670
50	490	118629		1425	1425,5	7200,8	13,41			659
Average			1404,63	1405,35	1405,63	5939,46	10,02	1,51%	0%	1403,47
#OPT								5	8	

Table 7.5: Tests carried out in order to obtain the maximum MST with $n = 50$.

Chapter 8

Conclusion

In this thesis, we have addressed two types of combinatorial optimization problems: *i)* substructure identification problems in graphs; and *ii)* combinatorial optimization problems with conflict constraints. In the context of the substructure identification problems we addresses the 2-Edge-Connected Minimum Branch Vertices Problem, the Collapsed k-Core Problem and the Cluster Deletion Problem.

In combinatorial optimization problems, introducing conflict constraints allows for managing real world's incompatibility situations. Their introduction to well-known problems makes these problems closer to the real case. The problems with conflict constraints studied in this thesis are: the Set Covering problem with Conflicts on Sets, the Max Flow with Conflicts and the Minimum Spanning Tree with Conflicts.

We developed a genetic algorithm to solve the 2ECMBV problem. This algorithm is based on a procedure able to find and remove useless edges from the feasible solution and some ad-hoc operators that increase the effectiveness of the approach by performing a broader exploration of the solution space. We tested the performance of the genetic algorithm on benchmark and new instances with respect to accuracy and running time. The computational results show that our algorithm is very effective in the benchmark instances where only in few cases it doesn't find the best/optimal solution. In the Hamiltonian instances, it is less effective, but the gap from the best/optimal solution remains low. Finally, GA results fast, with a computational time almost always lower than 1200 seconds.

Identifying the most critical users in terms of network engagement is a compelling topic in social network analysis. Users who leave a community potentially affect the cardinality of its k-core, i.e., the maximal induced subgraph of the network with a minimum degree of at least k. In this thesis, we presented different mathematical programming formulations of the Collapsed k-Core Problem, consisting in finding the b nodes of a graph, the removal of which leads to the k-core of minimal cardinality. We started with a time-indexed compact formulation that models the cascade effect after removing the b nodes. Then, we proposed two different bilevel

programming models of the problem. In both, the leader aims to minimize the cardinality of the k-core obtained by removing exactly b nodes. The follower wants to detect the k-core obtained after the decision of the leader on the b nodes to eliminate, i.e., to find the maximal subgraph of the new graph where all the nodes have a degree at least equal to k. The two formulations differ in the way the follower's problem is modelled. In the first one, the lower level is an ILP model solved through a Beneders-like decomposition approach. In the second bilevel formulation, the lower level is modelled through LP, which we dualized to end up with a single-level formulation. Preprocessing procedures and valid inequalities have been further introduced to enhance the proposed formulations. In order to evaluate the proposed formulations, we tested different existing instances, showing the superiority of the single-level reformulation of the second bilevel model. We further compared the approaches with the general purpose solver proposed in [FLMS17], which is outperformed by our problem-specific solution methods.

We present two exact ILP formulations and a heuristic algorithm, based on edge contraction operations to solve the Cluster Deletion problem. We tested and compared the proposed approaches on both artificial instances, generated by exploiting the Barabási–Albert model and benchmark biological networks. The performed experiments show that the proposed heuristic is very efficient and effective compared with the exact approaches, which provided the optimal solutions on the part of the instance set. Even when the algorithm does not identify the optimum, it still provides a high-value solution in a maximum of a few minutes. On a final note, since the observed relative gaps concerning the best found solutions are always small when the optimal solutions are available, we expect a similar behavior to apply also to the instances in which the best-known value is the one associated with the solution identified by the heuristic.

In the context of the combinatorial optimization problems with conflict constraints, in this thesis we addressed: the Set Covering problem with Conflicts on Sets, the Max Flow with Conflicts and the Minimum Spanning Tree with Conflicts.

The Set Covering Problem with Conflicts on Sets is a new variant of the set covering problem with conflicts among subsets, in which subsets are in conflict when they share a number of elements that exceeds a given threshold. Two subsets in conflict can belong to the same solution provided that a cost (proportional to the number of items that exceeds such threshold) is paid. We provide two mathematical formulations and a parallel variant of GRASP that exploits information sharing on the most demanding tasks. The proposed solution approach (tested with a time limit of 600 seconds) is highly effective, and often outperform Gurobi using the same number of processors and a larger amount of time (1 hour).

We investigated several heuristic approaches to solve the Maximum Flow Problem with Conflicts. On the one hand, a greedy algorithm has been designed and then enhanced according to the Carousel Greedy strategy. We also developed, a Kernel Search algorithm. Furthermore, we introduced Kernousel, i.e. a combination of Carousel Greedy and Kernel Search which results

in a fast and effective matheuristic. The proposed methods have been tested on benchmark instances and compared with the best known solutions from the literature. The computational tests show that exploiting the information, gathered by the Carousel Greedy to identify the set of promising variables for the Kernel Search, produces a highly effective solution framework.

Finally, we have dealt with a well-known variant of the Minimum Spanning Tree problem named Minimum Spanning Tree Problem with Conflicts. In order to address the problem, we have implemented a Kernel Search approach. We tested our approach making use of some benchmark instances already used in the literature. The obtained results show how KS lends itself to the resolution of this problem, improving the results obtained by some known heuristic methods and getting very close to the optimal solution. The study of this problem is still in progress and for future developments we would like to improve the efficiency and effectiveness of the KS approach, also considering other benchmark instances known in the literature. We also introduced a new bilevel variant of the problem named Minimum Spanning Tree Problem with Conflicts Placement. In this new variant we assume that the decision on the pairs of edges in conflict is made by a different entity w.r.t. the one deciding on the minimum spanning tree, and we model this scenario using Mixed-Integer Bilevel Linear Programming. We solved the problem using a general purpose solver for bilevel problems. The computational results have shown that the solver is able to solve only small and medium-sized. For future developments we would like to implement an ad hoc bilevel solution approach for the problem.

The study conducted in this thesis showed how the addressed problems can be tackled through the use of both classic tools of combinatorial optimization and new emerging techniques such as the Carousel Greedy algorithm, the Kernel Search matheuristic and the Kernousel algorithm obtained by combining the previous two approaches.

We conclude by specifying that there is still a lot of study to carry out regarding the validation of the new heuristic approach obtained by the combination of Kernel Search and Carousel Greedy but the preliminary results, reported in this work, are promising.

Bibliography

[ACMS02] E. Amaldi, A. Capone, F. Malucelli, and F. Signori. Umts radio planning: Optimizing base station configuration. 56:768–772, 2002. 88

[ALL11] A. Attar, H. Li, and V. C. M. Leung. Green last mile: How fiber-connected massively distributed antenna systems can save energy. *IEEE Wireles Communications*, 18(5):66–74, 2011. 87

[AMGS10] E. Angelelli, R. Mansini, and M. Grazia Speranza. Kernel search: A general heuristic for the multi-dimensional knapsack problem. *Computers and Operations Research*, 37(11):2017–2026, 2010. Cited By :84. 184

[AMS12] E. Angelelli, R. Mansini, and M.G. Speranza. Kernel search: A new heuristic framework for portfolio selection. *Computational Optimization and Applications*, 51:345–361, 2012. 136

[BA99] A. L. Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509 – 512, 1999. 77

[Bal65] E. Balas. An additive algorithm for solving linear programs with zero-one variables. *Operations Research*, 13(4):517–546, 1965. 181

[Bar09] A. L. Barabási. Scale-free networks: A decade and beyond. *Science*, 325(5939):412 – 413, 2009. 77

[BBBT09] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truss. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009. 70

[BBK11] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica (New York)*, 60(2):316–334, 2011. Cited By :64. 70

[BDVP15] F. Bonomo, G. Durán, and M. Valencia-Pabon. Complexity of the cluster deletion problem on subclasses of chordal graphs. *Theoretical Computer Science*, 600:59–69, 2015. 70

[Bea87] J. E. Beasley. An algorithm for set covering problem. *European Journal of Operational Research*, 31(1):85–93, 1987. Cited By :193. 105

[Bea90a] J. E. Beasley. Or-library. Website, 1990. 105

[Bea90b] J. E. Beasley. Or-library: Distributing test problems by electronic mail. *The Journal of the Operational Research Society*, 41(11):1069–1072, 1990. 105

[Bea92] J.E. Beasley. A lagrangean heuristic for set covering problems. pages 325–326, 1992. 105

[BGG14] N. Bilal, P. Galinier, and F. Guibault. An iterated-tabu-search heuristic for a variant of the partial set covering problem. *Journal of Heuristics*, 20(2):143–164, 2014. 87

[BHH22] C. Buchheim, D. Henke, and F. Hommelsheim. On the complexity of the bilevel minimum spanning tree problem. *Networks*, 80(3):338–355, 2022. 136

[BKL+15] K. Bhawalkar, J. Kleinberg, K. Lewi, T. Roughgarden, and A. Sharma. Preventing unraveling in social networks: the anchored k-core problem. *SIAM Journal on Discrete Mathematics*, 29(3):1452–1475, 2015. 40, 42

[BP76] E. Balas and M. W. Padberg. Set partitioning: A survey. *SIAM review*, 18(4):710–760, 1976. 86

[BPOR19] B. Barros, R. Pinheiro, L. Ochi, and G. Ramos. A grasp approach for the minimum spanning tree under conflict constraints. In *Anais do XVI Encontro Nacional de Inteligência Artificial e Computacional*, pages 166–177. SBC, 2019. 135

[BR01] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35:268–308, 01 2001. 177

[BZ03] V. Batagelj and M. Zaversnik. An O(m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003. 41, 42

[Bö12] S. Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012. 70

[cAA20] Z. Şuvak, İ. K. Altınel, and N. Aras. Exact solution algorithms for the maximum flow problem with additional conflict constraints. *European Journal of Operational Research*, 287(2):410–437, 2020. 15, 114, 115, 129

[CCCG20] F. Carrabs, C. Cerrone, R. Cerulli, and B. Golden. An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 32(4):1030–1048, 2020. Cited By :8. 130, 176

[CCG17] C. Cerrone, R. Cerulli, and B. Golden. Carousel greedy: A generalized greedy algorithm with applications in optimization. *Computers and Operations Research*, 85:97–112, 2017. Cited By :43. 122, 175

[CCGG13] 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. 17

[CCM+17] M. Colombi, Á. Corberán, R. Mansini, I. Plana, and J. M. Sanchis. The directed profitable rural postman problem with incompatibility constraints. *European Journal of Operational Research*, 261(2):549–562, 2017. 14

[CCP19] F. Carrabs, C. Cerrone, and R. Pentangelo. A multiethnic genetic approach for the minimum conflict weighted spanning tree problem. *Networks*, 74(2):134–147, 2019. 14, 135

[CCPR21a] F. Carrabs, R. Cerulli, R. Pentangelo, and A. Raiconi. Minimum spanning tree with conflicting edge pairs: a branch-and-cut approach. *Annals of Operations Research*, 298(1):65–78, 2021. 10, 135, 142, 143, 145, 146, 147, 148, 149

[CCPR21b] F. Carrabs, R. Cerulli, R. Pentangelo, and A. Raiconi. Minimum spanning tree with conflicting edge pairs: a branch-and-cut approach. *Annals of Operations Research*, 298(1):65–78, 2021. 14

[CDF+11] J. Cardinal, E. D. Demaine, S. Fiorini, G. Joret, S. Langerman, I. Newman, and O. Weimann. The stackelberg minimum spanning tree game. *Algorithmica*, 59(2):129–144, 2011. 136

[CDIP22] R. Cerulli, C. D'Ambrosio, A. Iossa, and F. Palmieri. Maximum network lifetime problem with time slots and coverage constraints: heuristic approaches. *Journal of Supercomputing*, 78(1):1330–1355, 2022. Cited By :1. 176

[CDLMR02] G Cerri, R De Leo, D Micheli, and P Russo. Reduction of the electromagnetic pollution in mobile communication systems by an optimized location of radio base stations. In *Proc. XXVIIth Gen. Assembly URSI*, 2002. 87

[CDP+22] G. Capobianco, C. D'Ambrosio, L. Pavone, A. Raiconi, G. Vitale, and F. Sebastiano. A hybrid metaheuristic for the knapsack problem with forfeits. *Soft Computing*, 26(2):749–762, 2022. Cited By :2. 176

[CDR19] C. Cerrone, C. D'Ambrosio, and A. Raiconi. Heuristics for the strong generalized minimum label spanning tree problem. *Networks*, 74(2):148–160, 2019. Cited By :8. 176

[CDRV20] R. Cerulli, C. D'Ambrosio, A. Raiconi, and G. Vitale. *The knapsack problem with forfeits*, volume 12176 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2020. Cited By :1. 176

[Cer21] M. Cerulli. Bilevel optimization and applications. *Operations Research [cs.RO]. Institut Polytechnique de Paris, 2021. English. NNT : 2021IPPAX108. tel-03587548*, 2021. 41

[CFG13] R. Chitnis, F. V. Fomin, and P. A. Golovach. Preventing unraveling in social networks gets harder. pages 1085–1091, 2013. 42

[CFSS21] S. Coniglio, F. Furini, and P. San Segundo. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, 289(2):435–455, 2021. 14

[CG21a] F. Carrabs and M. Gaudioso. A lagrangian approach for the minimum spanning tree problem with conflicting edge pairs. *Networks*, 78(1):32–45, 2021. 14

[CG21b] F. Carrabs and M. Gaudioso. A lagrangian approach for the minimum spanning tree problem with conflicting edge pairs. *Networks*, 78(1):32–45, 2021. 135, 145, 146

[CGDC18] C. Cerrone, M. Gentili, C. D'Ambrosio, and R. Cerulli. *An Efficient and Simple Approach to Solve a Distribution Problem*, volume 1 of *AIRO Springer Series*. 2018. 130, 176

[CGI09] R. Cerulli, M. Gentili, and A. Iossa. Bounded-degree spanning tree problems: models and new algorithms. *Computational Optimization and Applications*, 42:353, 2009. 18

[CLRS22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms, 4th edition*. The MIT Press, 2022. 27, 118, 120

[CS07] P. Colson, B.and Marcotte and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153:235–256, 2007. 41

[CTF00] A. Caprara, P. Toth, and M. Fischetti. Algorithms for the set covering problem. *Annals of Operations Research*, 98(1):353–371, 2000. 86

[DAE+07] A. Dessmark, Lingas A., Lundell E., Persson M., and Jansson J. On the approximability of maximum and minimum edge clique partition problems. *International Journal of Foundations of Computer Science*, 18(2):217–226, 2007. 70

[Dem02] S. Dempe. *Foundations of Bi-Level Programming*. Nonconvex Optimization and Its Applications. Springer US, 1 edition, 2002. 41

[DFJ54] G. Dantzig, R. Fulkersonì, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954. 181

[DFJ58] G. Dantzig, R. Fulkersonì, and S. Johnson. *Bulletin of the American Mathematical Society*, (64):275–278, 1958. 181

[DMM97] M. Dell'Amico, F. Maffioli, and S. Martello. *Annotated bibliographies in combinatorial optimization*. Wiley, 1997. 86

[DPST09] A. Darmann, U. Pferschy, editor="Rossi F. Schauer, J.", and A. Tsoukias. Determining a minimum spanning tree with disjunctive constraints. In *Algorithmic Decision Theory*, pages 414–423, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 135

[DPSW11] 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. Cited By :61. 14, 135

[EFL11] L. Epstein, L. M. Favrholdt, and A. Levin. Online variable-sized bin packing with conflicts. *Discrete Optimization*, 8(2):333–343, 2011. 14

[Eki21] A. Ekici. Bin packing problem with conflicts and item fragmentation. *Computers and Operations Research*, 126, 2021. 14

[ET76] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, 1976. 21

[FBJ04] A. Figueroa, J. Borneman, and T. Jiang. Clustering binary fingerprint vectors with missing values for dna array data analysis. *Journal of Computational Biology*, 11(5):887–901, 2004. 69

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics.*, 8:399–404, 1956. 13

[FLMP20] F. Furini, I. Ljubić, E. Malaguti, and P. Paronuzzi. On integer and bilevel formulations for the k-vertex cut problem. *Mathematical Programming Computation*, 12:133–164, 2020. 40

[FLMS17] M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl. A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65(6):1615–1637, 2017. 14, 42, 62, 63, 145, 149, 156

[FLMS19] M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl. Interdiction games and monotonicity, with application to knapsack problems. *INFORMS Journal on Computing*, 31(2):390–410, 2019. Cited By :31. 53

[FLSSM19] F. Furini, I. Ljubić, P. San Segundo, and S. Martin. The maximum clique interdiction problem. *European Journal of Operational Research*, 277(1):112–127, 2019. 40

[FLSSZ21a] F.. Furini, I. Ljubić, P. San Segundo, and Y. Zhao. A branch-and-cut algorithm for the edge interdiction clique problem. *European Journal of Operational Research*, 294(1):54–69, 2021. 40

[FLSSZ21b] F. Furini, I. Ljubić, P. San Segundo, and Y. Zhao. A branch-and-cut algorithm for the edge interdiction clique problem. *European Journal of Operational Research*, 294(1):54–69, 2021. Cited By :7. 40

[FR89] T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989. 178

[FR95] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995. 178

[FSO99] N. Frederickson and R. Solis-Oba. Increasing the weight of minimum spanning trees. *Journal of Algorithms*, 33(2):244–266, 1999. 136

[Gas02] E. Gassner. Maximal spanning tree problems with a hierarchy of two decision makers, 2002. Diploma Thesis In German, Graz University of Thechnology. 136

[Gas09] E. Gassner. The computational complexity of continuous-discrete bilevel network problems. 2009. 136

[GHN13] Y. Gao, D. R. Hare, and J. Nastos. The cluster deletion problem for cographs. *Discrete Mathematics*, 313(23):2763–2771, 2013. 70

[GHSV02] L. Gargano, P. Hell, L. Stacho, and U. Vaccaro. Spanning trees with bounded number of branch vertices. *International Colloquium on Automata, Languages, and Programming, Springer Berlin Heidelberg*, 2380:355–365, 2002. 13, 17, 19

[GI91] Z. Galil and G. F. Italiano. Reducing edge connectivity to vertex connectivity. *SIGACT News*, 22(1):57–61, March 1991. 21

[GN02] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002. 64

[GS86] A. M. H. Gerards and A. Schrijver. Matrices with the edmonds—johnson property. *Combinatorica*, 6(4):365–379, 1986. 143

[GVPP21] C. P. Gillen, A. Veremyev, O. A. Prokopyev, and E. L. Pasiliao. Fortification against cascade propagation under uncertainty. *INFORMS Journal on Computing*, 33(4):1481–1499, 2021. 46

[GW97] T. Grossman and A. Wool. Computational experience with approximation algorithms for the set covering problem. *European Journal of Operational Research*, 101(1):81–92, 1997. 105

[GWC+13] W. Guo, S. Wang, X. Chu, J. Zhang, J. Chen, and H. Song. Automated small-cell deployment for heterogeneous cellular networks. *IEEE Communications Magazine*, 51(5):46–53, 2013. 87

[HJ97] P. Hansen and B. Jaumard. Cluster analysis and mathematical programming. *Mathematical Programming, Series B*, 79(1-3):191–215, 1997. 69

[HM07] M. Hifi and M. Michrafy. Reduction strategies and exact algorithms for the disjunctively constrained knapsack problem. *Computers & operations research*, 34(9):2657–2673, 2007. 14

[HSS08] A Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008. 33

[KLLS21] T. Kleinert, M. Labbé, I. Ljubić, and M. Schmidt. A survey on mixed-integer programming techniques in bilevel optimization. *EURO Journal on Computational Optimization*, 9, 2021. Cited By :16. 41, 53

[KLM13] M. M. Kanté, C. Laforest, and B. Momège. Trees in graphs with conflict edges or forbidden transitions. 7876 LNCS:343–354, 2013. Cited By :16. 135

[Knu93] D. E Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley Educational, Boston, MA, November 1993. 64

[KPS11] J. Könemann, O. Parekh, and D. Segev. A unified approach to approximating partial covering problems. *Algorithmica*, 59(4):489–509, 2011. 87

[Kre99] V. Krebs. The Social Life of Books Visualizing Communities of Interest via Purchase Patterns on the WWW, 1999. http://orgnet.com/booknet.html [Accessed: 27.10.2022]. 64

[KT93] S. Khuller and R. Thurimella. Approximation algorithms for graph augmentation. *Journal of Algorithms*, 14(2):214–225, 1993. 21

[KU11] C. Komusiewicz and J. Uhlmann. Alternative parameterizations for cluster editing. 6543 LNCS:344–355, 2011. 70

[KU12] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. 70

[KV98] A. Krause and M. Vingron. A set-theoretic approach to database searching and clustering. *Bioinformatics*, 14(5):430–438, 1998. Cited By :43. 69

[Lau19] F. Laureana. *Polyhedral Analysis and Branch and Cut Algorithms for Some NP-hard Spanning Subgraph Problems*. PhD thesis, University of Salerno, 2019. 13, 14, 22, 23, 33, 34

[LD10] A. H. Land and A. G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer, 2010. 181

[LIDLPC+16] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. 33

[LL93] G. Laporte and F. V. Louveaux. The integer l-shaped method for stochastic integer programs with complete recourse. *Operations Research Letters*, 13(3):133–142, 1993. Cited By :430. 59

[LLM+22] M. Leitner, I. Ljubić, M. Monaci, M. Sinnl, and K. Tanınmış. An exact method for binary fortification games. *European Journal of Operational Research*, 2022. 53

[LMS21] J. Luo, H. Molter, and O. Suchý. A parameterized complexity view on collapsing k-cores. *Theory of Computing Systems*, 65(8):1243–1282, 2021. 42

[LSB+03] D. Lusseau, K. Schneider, O J Boisseau, P. Haase, E. Slooten, and S M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations - can geographic isolation explain this unique trait? *Behavioral Ecology and Sociobiology*, 54:396–405, 2003. 64

[LSER+20] R. Laishram, A. E. Sariyüce, T. Eliassi-Rad, A. Pinar, and S. Soundarajan. Residual core maximization: An efficient algorithm for maximizing the size of the k-core. pages 325–333, 2020. Cited By :7. 42

[Mar15] A. Marín. Exact and heuristic solutions for the minimum number of branch vertices spanning tree problem. *European Journal of Operational Research*, 245(3):680–689, 2015. 18

[MB83] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983. 42

[MGKP09] B. Mamalis, D. Gavalas, C. Konstantopoulos, and G. Pantziou. *Clustering in wireless sensor networks*, pages 323–354. RFID and Sensor Networks: Architectures, Protocols, Security, and Integrations. 2009. 69

[MGPV20] F. D. Malliaros, C. Giatsidis, A. N. Papadopoulos, and M. Vazirgiannis. The core decomposition of networks: theory, algorithms and applications. *VLDB Journal*, 29(1):61–92, 2020. 42

[MN19] S. Malek and W. Naanaa. A new polynomial algorithm for cluster deletion problem. 790:76–88, 2019. 70

[New01] M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences of the United States of America*, 98(2):404–409, 2001. Cited By :3418. 64

[New06] M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74:036104, Sep 2006. 64

[NSS01] A. Natanzon, R. Shamir, and R. Sharan. Complexity classification of some edge modification problems. *Discrete Applied Mathematics*, 113(1):109–128, 2001. Cited By :136. 69

[ÖKA18] T. Öncan and İ. Kuban Altınel. A branch-and-bound algorithm for the minimum cost bipartite perfect matching problem with conflict pair constraints. *Electronic Notes in Discrete Mathematics*, 64:5–14, 2018. 14

[OZP13] T. Öncan, 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. 14

[PR91] M. Padberg and G. Rinaldi. Branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33(1):60–100, 1991. Cited By :614. 181

[PS09] U. Pferschy and J. Schauer. The knapsack problem with conflict graphs. *Journal of Graph Algorithms and Applications*, 13:233–249, 2009. 14

[PS13a] U. Pferschy and J. Schauer. The maximum flow problem with disjunctive constraints. *Journal of Combinatorial Optimization*, 26(1):109–119, 2013. 15

[PS13b] U. Pferschy and J. Schauer. The maximum flow problem with disjunctive constraints. *Journal of Combinatorial Optimization*, 26(1):109–119, 2013. 114

[PSS+02] P. Pipenbacher, A. Schliep, S. Schneckener, A. Schönhuth, D. Schomburg, and R. Schrader. Proclust: Improved clustering of protein sequences with an extended graph-based approach. *Bioinformatics*, 18(SUPPL. 2):S182–S191, 2002. 69

[PW83] M. W Padberg and L. A. Wolsey. Trees and cuts. 75:511–517, 1983. 143

[RA15] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. 6:4292–4293, 2015. Cited By :721. 77, 83

[RRD04] Shamir R., Sharan R., and Tsur D. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1):173–182, 2004. 69, 70

[RTJ93] K. A. Ravindra, L. M. Thomas, and B. O. James. *Network Flows, theory, algorithms, and applications*. Prentice-Hall, New Jersey, 1993. 18

[Sch02] A. Schrijver. On the history of the transportation and maximum flow problems. *Mathematical Programming, Series B*, 91(3):437–445, 2002. Cited By :117. 13

[Sch13] J. M. Schmidt. A simple test on 2-vertex- and 2-edge-connectivity. *Information Processing Letters*, 113(7):241 – 244, 2013. 21

[Sei83] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269–287, 1983. Cited By :1184. 42

[SLC17] S. Silvestri, G. Laporte, and R. Cerulli. A branch-and-cut algorithm for the minimum branch vertices spanning tree problem. *Computers and Operations Research*, 81:322–332, 2017. 17

[SP16] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *Proceedings of the Very Large Data Bases Endowment*, 10(3):97–108, 2016. 42

[SPR22] X. Shi, O. A. Prokopyev, and T. K. Ralphs. Mixed integer bilevel optimization with a k-optimal follower: a hierarchy of bounds. *Mathematical Programming Computation*, pages 1–51, 2022. 136

[SS20] J. C. Smith and Y. Song. A survey of network interdiction models and algorithms. *European Journal of Operational Research*, 283(3):797–811, 2020. Cited By :55. 53

[SSR+14] R. M. A. Silva, D. M. Silva, M. G. C. Resende, G. R. Mateus, J. F. Gonçalves, and P. Festa. An edge-swap heuristic for generating spanning trees with minimum number of branch vertices. *Optimization Letters*, 8(4):1225–1243, 2014. 18

[STM17] B. Saminathan, I. Tamilarasan, and M. Murugappan. Energy and electromagnetic pollution considerations in arof-based multi-operator multi-service systems. *Photonic Network Communications*, 34(2):221–240, 2017. 87

[SU15] P. Samer and S. Urrutia. A branch and cut algorithm for minimum spanning trees under conflict constraints. *Optimization Letters*, 9(1):41–55, 2015. 135

[SV13] R. Sadykov and F. Vanderbeck. Bin packing with conflicts: A generic branch-and-price algorithm. *INFORMS Journal on Computing*, 25(2):244–255, 2013. 14

[SZP19] X. Shi, B. Zeng, and O. A. Prokopyev. On bilevel minimum and bottleneck spanning tree problems. *Networks*, 74(3):251–273, 2019. Cited By :2. 136

[Tar72] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972. 21

[Tar74] R. E. Tarjan. A note on finding the bridges of a graph. *Information Processing Letters*, 2(6):160–161, 1974. 19, 21

[Uhl11] J. Uhlmann. Multivariate algorithmics in biological data analysis. 2011. 70

[Uni04] University of Oregon. Route views archive project, 2004. http://routeviews.org/ [Accessed: 27.10.2022]. 64

[VC94] L. N. Vicente and P. H. Calamai. Bilevel and multilevel programming: A bibliography review. *Journal of Global Optimization*, 5(3):291–306, 1994. 41

[Vik17] P. A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. pages 261–265, 2017. Cited By :177. 177

[WJG13] X. Wang, Z. Jiang, and S. Gao. An enhanced difference method for multi-objective model of cellular base station antenna configurations. 5:361–366, 2013. 88

[Woo11] R. K. Wood. Bilevel network interdiction models: Formulations and solutions. 2011. 53

[WS98] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world9 networks. *Nature*, 393(6684):440–442, 1998. Cited By :30370. 64

[YLR+19] S. Yu, Y. Liu, J. Ren, H. D. Bedru, T. M. Bekele, L. Wan, and F. Xia. Mining key scholars via collapsed core and truss. pages 305–308, 2019. Cited By :1. 43

[Zac77] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977. 64

[ZCWL18] W. Zhu, C. Chen, X. Wang, and X. Lin. K-core minimization: An edge manipulation approach. pages 1667–1670, 2018. 43

[ZKP11] 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. Cited By :46. 135

[ZY20] J. Zhang and Y. Yang. Research on collapsed (α, k)-NP-community on signed graph. 1486, 2020. 43

[ZZQ+17] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31(1), 2017. 40, 42, 43

[ZZZ+16] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. Olak: An efficient algorithm to prevent unraveling in social networks. *Proceedings of the VLDB Endowment*, 10(6):649–660, 2016. Cited By :51. 42

Appendix A

Recall on Solution Approaches for Optimization Problems

A.1 Basic Concepts: Combinatorial Optimization

Optimization is the study of maximization/minimization of functions (objective functions) whose variables are constrained to satisfy specific conditions (constraints). In the following, we will refer to minimization problems. Very often, optimization is linked to combinatorial analysis, which is the mathematics of discrete structured problems. A combinatorial optimization problem (COP) consists in finding out the value of a certain combination of variables that optimizes a given function under various constraints.

Given a function $f : \mathbb{R}^n \to R$ and $X \subseteq \mathbb{R}^n$, optimization problem can be formulated as:

$$\min f(x)$$

$$s.t.$$

$$x \in X$$

An optimization problem consists in determining, if it exists, a global minimum point of the function f among the points of the set X. The function f is called the objective function of the problem and describes the objective to be pursued, x is the vector of the decision variables of the problem and represents a possible solution, while X is the set of feasible solutions, and is also called feasible region. When the set of feasible solutions of an optimization problem is expressed through a system of equations and inequalities, the problem is referred to as Mathematical Programming (PM) problem. A PM problem is linear when both the objective function f and the feasible set of constraints X are expressed in terms of linear relations (equalities and inequalities). A typical linear MP model has the following form:

$$\min f(x)$$

$$s.t.$$

$$g_i(x) \geq b_i \quad i = 1, \ldots, m$$

$g_i(x)$ is the i-th constraint of the system, and b_i is the i-th constant term. It can be rewritten in compact form using matrix notation:

$$\min_{x \in \mathbb{R}^n} c^T x \tag{A.1a}$$

$$s.t.\ Ax = b \tag{A.1b}$$

$$x \geq 0 \tag{A.1c}$$

where $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. The equations A.1b are called constraints of the problem and define the so-called region X of feasible solutions, constraining the domain within the variables of the problem can assume values. The model A.1 is the so-called standard form of linear optimization problem (LP). It can be shown that every LP problem can be written in standard form by introducing suitable variable splittings and/or slack variables. The objective function $f : X \to \mathbb{R}$ allows to orient the search within the set X, associating a value to each potential solution, which can represent a cost to be minimized or, alternatively, a profit to be maximized. We are interested in finding a *global optimum*, i.e. a solution $x \in X$ such that $f(x) \leq f(y); \forall y \in X$. On the other hand, some solutions could turn out to be *local optima*, i.e. have a better value only than the solutions belonging to a well-defined neighborhood of x. Furthermore, if the variables of the problem assume continuous values $x \in \mathbb{R}^n$ we speak of continuous Linear Programming (LP). When the variables of the problem are constrained to assume only integer values, the problem is called Integer Linear Programming (ILP). However, if some decision variables are not discrete, the problem is known as a Mixed Integer Linear Programming problem (MILP). Although ILP and MILP formulations are generally harder to tackle, they achieved considerable success over time, thanks of their properties and the existence of effective methods for their solution, both heuristic and exact.

An algorithm is said to solve an LP problem if it is capable of correctly determining whether the given problem has an empty feasible region or is unbounded or, if neither of these two cases is verified, is capable of identifying an optimal solution. There exist many algorithms for solving Linear Programming problems. The most used is the Simplex Method, which was the first practical algorithm for solving Linear Programming problems and is still the most used and one of the most efficient in practice, despite its computational complexity that is not polynomial unlike other approaches.
It is not always possible to model a problem through constraints and linear objective functions. It can be necessary that some of the constraints or the objective function are nonlinear. The general form of such a nonlinear optimization problem (NLP) reads

$$\min_{x \in \mathbb{R}^n} f(x) \tag{A.2a}$$

$$s.t.\ g_i(x) \geq 0,\ \ i \in I = \{\, 1, \ldots, m \,\}, \tag{A.2b}$$

$$h_i(x) = 0,\ \ j \in J = \{\, 1, \ldots, p \,\} \tag{A.2c}$$

We assume that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ as well as the constraint functions $g_i : \mathbb{R}^n \to \mathbb{R}$, $i \in I$ and $h_j : \mathbb{R}^n \to \mathbb{R}$, $j \in J$, are continuously differentiable. The feasible set is denoted by $\mathcal{F}$.

Solution methods for Combinatorial Optimization Problems (COPs) fall into two classes: *heuristic* and *exact* methods.

In the following the various solution approaches studied to address the problems covered in this thesis are described, which are: *Heuristic approaches* in Section A.2, *Meta-Heuristic approaches* in Section A.3, *Exact approaches* in Section A.4 and *Math-Heuristic approaches* in Section A.5.

A.2 Heuristic approaches

Since COPs are usually NP-hard, it is often necessary to develop heuristic algorithms, i.e. algorithms which do not guarantee obtaining the optimal solution, but in general are able to provide a good feasible solution for the problem. Normally heuristic algorithms have low complexity, but in some cases, for large problems and complex structure, it may be necessary to develop sophisticated and highly complex heuristic algorithms. Furthermore, it is possible, in general, for a heuristic algorithm to fail and not be able to determine any feasible solution to the problem, without being able to prove that none exists. Designing an effective heuristic algorithms requires a careful analysis of the problem to be solved aimed at identifying its structure, ie the specific useful characteristics, and a good knowledge of the main algorithmic techniques available. In fact, even if each problem has its specific characteristics, there are a number of general techniques that can be applied, in different ways, to many problems, producing well-defined classes of optimization algorithms. An example of a heuristic algorithm is the greedy algorithm, which represents one of the simplest algorithms in the field of optimization. Another kind of sophisticated algorithm is the local search algorithm, which allows to escape from local optima.

A large part of heuristic algorithms fall into the category of constructive algorithms (CA). CAs start from an empty solution and iteratively determine the new elements to add in the solution until a complete solution is obtained. Greedy algorithms are part of this category.
Greedy algorithms determine the solution through a sequence of locally optimal decisions, without ever going back and changing the decisions made. These algorithms are easy to implement and are characterized by a remarkable computational efficiency, but, except for some important cases, in general they do not guarantee the optimality, and sometimes not even the feasibility, of

the solution found. The definition we have given of greedy algorithm is very general, and therefore algorithms that appear very different from each other can be traced back to this category. However, it is possible to define a basic schema described by algorithm 23. We denote with E the set of all possible choices and we denote with F the set of feasible solutions of the problem.

Algorithm 23: Greedy

```
   Input: E, F
1  S = ∅
2  Q = E
3  while |Q| ≠ 0 do
4  |   e = Q.pop()
5  |   if S ∪ {e} ∈ F then
6  |   |   S = S ∪ {e}
7  return S
```

In the algorithm 23, S is the set of elements of E that have been inserted into the current (partial) solution, and Q is a priority queue, a data structure that allows access to the elements contained in it according to a certain order. Q is initially populated with elements of the problem E; for example, if the items in E have a cost associated with them, the priority queue may choose the cheapest items first. The *pop* method of the queue returns the next element to be extracted following the greedy rule, in this case it is simply an ordering of the elements indicating which one is more attractive. If the algorithm is not able to construct a feasible solution it fails, i.e. it is unable to determine a feasible solution of the problem.

Another well-known heuristic algorithm is Local Search (LS). The LS algorithms are based on a simple and intuitive idea: given a feasible solution, the solutions *close to it* are examined in search of a better solution, i.e. with a better value of the objective function. If such a solution is found, it becomes the current solution and the procedure is iterated, otherwise (i.e. when none of the nearby solutions is better than the current one) the algorithm terminates having determined a local optimum of the feasible set.

Of fundamental importance for this type of algorithm is the definition of a neighborhood function. In general, given the feasible set F of the problem under consideration, we can define a function $N : F \to 2^F$, where the set $N(x)$ is called neighborhood of x, and contains the solutions considered close to x. Given a neighborhood function, a local search algorithm can be

schematized as follows:

Algorithm 24: Local Search

```
   Input: F, x
 1 x = AmmissibleSolution(F)
 2 repeat
 3 |   N(x) ←compute neighborhood of x
 4 |   y = best solution in N(x)
 5 |   if f(y) < f(x) then
 6 |   |   x = y
 7 |   else
 8 |   |   break
 9 until True
```

Algorithm 24 shows that the method is extremely general and can be applied to solve very different problems. The salient steps of the algorithm concern: *i)* the determination of the initial solution, which can be performed through a greedy algorithm; and *ii)* the definition of a neighborhood of the current solution x, also called *exploration criterion*. At each step of the algorithm a restricted optimization problem is solved in the considered neighborhood. In general, the solution determined by the local search algorithm is not optimal for the problem, but only a local optimum related to the chosen neighborhood function N. A neighborhood function is called an exact neighborhood function if for a given problem the local search algorithm is able to provide the optimal solution for each instance of the problem however the starting point is chosen.

A particular local search algorithm is the Hill Climbing algorithm, it is a heuristic algorithm that is often applied in the field of Artificial Intelligence (AI). It is generally assumed that the problem is maximization of a given objective function. The Hill Climbing algorithm iteratively increases the value of the current solution until it reaches a peak solution, continuously moving upward (increasing) until the best local solution is attained (local optimum). It begins with a non-optimal state (the hill's base) and upgrades this state until a certain precondition is met. A heuristic function is used as the basis for this precondition. The process of continuous improvement of the current state of iteration can be compared to climbing. This explains why the algorithm is named hill-climbing. A hill-climbing algorithms objective is to attain an optimal state that is an upgrade of the existing state. When the current state is improved, the algorithm will perform further incremental changes to the improved state. This process will continue until a peak solution is achieved. The peak state cannot undergo further improvements.

There are two main regions in which a hill-climbing algorithm cannot reach a global optimum: local optimum and plateau. In an local optimum point, the state of the neighbors have lower values than the current state. This will lead to the hill-climbing process's termination, even though this is not the best possible solution. This problem can be solved by using momentum. Momentum allows the hill climbing algorithm to take huge steps that will cause it to exceed the local optimum. In a plateau region, the values attained by the neighboring states are the

same. This makes it difficult for the algorithm to choose the best direction. This challenge can be overcome by taking a huge jump that will lead you to a non-plateau space.

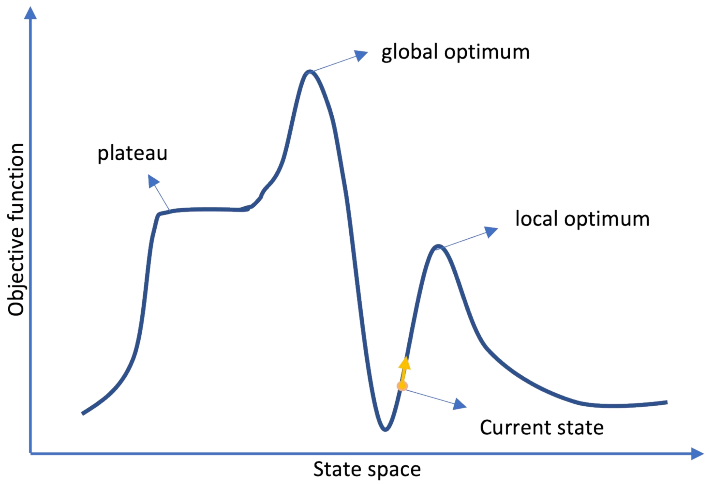


Figure A.1: Scheme of the hill climbing algorithm.

Getting stuck in a local optimum is a typical problem of heuristic algorithms, to overcome this type of problem metaheuristic algorithms have been developed, some of which used in this thesis are described in the Section A.3.

A.2.1 Carousel Greedy

In this section we describe the Carousel Greedy approach which is an enhanced greedy algorithm proposed by Cerrone et.al. [CCG17], with the aim of overcoming the classic weaknesses of greedy algorithms while maintaining their efficiency. The main idea of the CG approach is to start from a feasible solution, generated through a greedy algorithm, and modify it by replacing the older choices made by the greedy algorithm with new choices that produce a new feasible solution. Since the choices made in the initial and final phases of the starting greedy algorithm are the most constrained ones, this process naturally enhances such procedure, by actually extending the exploration phase, with a reduced computational overhead. It allows to resolve large instances in a short time, as it still falls into the category of heuristic algorithms and not into the category of metaheuristic algorithms, which in order to escape from a local optimum require a greater computational effort. It can be combined with other approaches to create or support new metaheuristics.

Several well-known combinatorial optimization problems, like the minimum label spanning tree, the minimum vertex cover, the maximum independent set, and the minimum weight vertex cover problems served as first case study for the CG framework [CCG17]. By now, the resulting

scheme has been applied to a variety of problems, including distribution problems [CGDC18], the strong generalized minimum label spanning tree problem [CDR19], the knapsack problem with forfeits [CDRV20, CDP+22], the close-enough traveling salesman problem [CCCG20], and the maximum network lifetime problem with time-slots [CDIP22]. CG requires as input: an initial solution provided by a Greedy algorithm and two parameters, α and β, where α is an integer that determines the number of iterations of the algorithm and β is a percentage that allows us to cut a part of the initial greedy solution. The main step of carousel greedy algorithm are the following:

- Use a greedy algorithm to generate a partial solution;
- Apply the same greedy algorithm starting from the partial solution in a deterministic manner.
- Use the greedy algorithm to complete the solution, obtaining a feasible one.

Figure A.2 shows the Greedy Carousel execution scheme. In the first line a starting solution $S = \{e_1, \ldots, e_{|S|}\}$ (where e_i indicates the i-th element contained in the solution S), is built through a constructive algorithm. Then the CG removes the $\beta\%$ from the elements of the final solution, obtaining what is called *Carousel Start*, the real starting point of the CG. This represents the first phase of the CG algorithm. The algorithm iterates $\alpha|S|$ times where $|S|$ is the size of the initial solution. At each iteration, the CG removes the oldest element, and replaces it with a new one, using the rule defined by the starting greedy to choose the next move. This replacement policy is applied because, when the greedy algorithm starts its computation, it has few information available and the initial choices are almost never the best. By deleting the oldest choice, we give the algorithm the chance to replace a wrong choice with one that at the current iteration seems more advantageous. Indeed, if the removed element is still the best, the CG choose it again, inserting it into solution. Once arrived at the last iteration, the algorithm completes the solution by applying the starting greedy algorithm, letting it insert as many elements as possible.

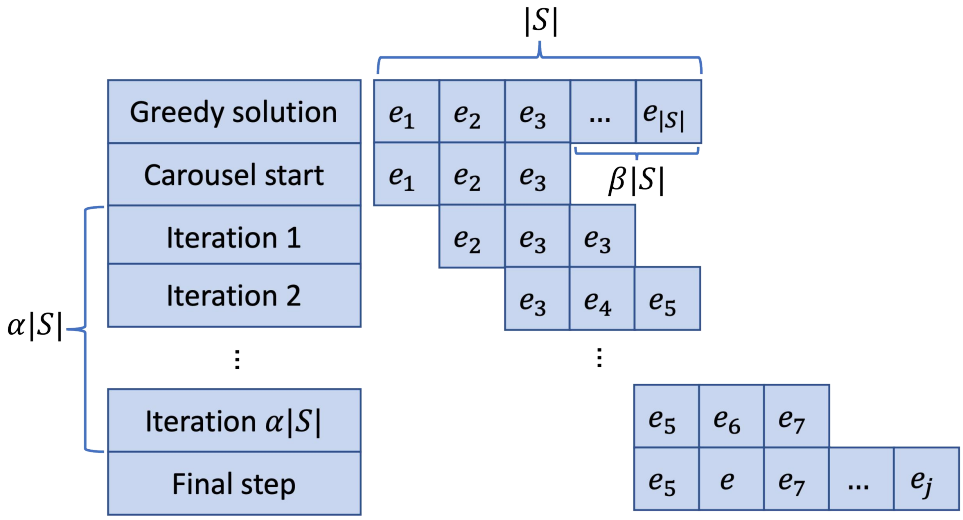


Figure A.2: Scheme of the Carousel Greedy algorithm.

A.3 Metaheuristic approaches

Even using large neighborhoods in Local Search approaches, described in the previous section, in many combinatorial optimization problems one may not be able to identify the optimal solution. The local search algorithms, for example, stop after having identified a local optimum with respect to the neighborhood function used, as they are unable to find "moves" that generate better solutions than the current one. If the current solution identified by a heuristic approach is not a global optimum, then the problem of trying to determine a different, and possibly better, local optimum arises. In this section, we will briefly discuss two possible strategies that allow to do this, used to solve the problems addressed in this thesis. These strategies are called metaheuristics because they are not specific algorithms for a given problem, but general methods that can be applied to try to improve the performance of many different local search algorithms.

There exist different types of metahaeristics, that can be grouped into two macro-categories. The first category contains approaches that are based on a certain type of search strategy [BR01]. They consist of a simple improvement of a local search algorithm. Some metaheuristics of this type are Simulated Annealing, Tabu Search, Iterated Local Search, GRASP, etc. These metaheuristics can be classified as local search-based metaheuristics.

The second category concerns global search metaheuristics which are based on the generation of a population of solutions [Vik17]. Such metaheuristics include: ant colony optimization, evolutionary computation, particle swarm optimization, etc. This approaches are also based on a search strategy but which differs from the classic one used in the context of local search.

In the following, we describe two metaheuristics used to solve two of the problems addressed

in this thesis. The first metaheuristic belongs to the first category and is described in Section A.3.1, while the second one belongs to the second category and is described in the Section A.3.2.

A.3.1 GRASP

The quality of the local optimum determined by a local search algorithm depends on two factors: *i)* the neighborhood used; *ii)* the initial feasible solution from which the search starts. In the Section A.2 we assumed that the initial solution was determined by some heuristic, for example a greedy algorithm. It is possible to define more than one greedy algorithm for the same problem, and in many cases the algorithms are also very similar, differing only in the order in which some choices are made, so it is reasonable to think of having more than one available algorithm capable of producing initial solutions. In this case, the solutions produced will normally be different; moreover, it is said that the local optimum determined by executing the local search algorithm starting from the best of the solutions thus obtained is not necessarily the best of the local optima obtainable by executing the local search algorithm starting from each of the solutions separately. All this suggests an obvious extension of the local search algorithm: generate several initial solutions, compute the neighborhood of each solution and execute the local search algorithm starting from each of them, then select the best of the solutions thus obtained. This procedure is particularly attractive when it is possible, typically through the use of randomized techniques, to easily generate an arbitrarily large set of different initial solutions.
The combination of a local search algorithm and a randomized heuristic is called the *multistart method*. When the randomized heuristic is of the greedy type, it is called a Greedy Randomized Adaptive Search Procedure (GRASP) [FR89, FR95]. While other metaheuristic algorithms such as genetic algorithms use strategies with great emphasis on local search, the GRASP approach is considered constructive because it is focused on the generation of a better quality initial solution in order to use local search only for small improvements.

The algorithms 25 and 26 describe, respectively, the first phase of GRASP (constructive greedy) and the second phase of GRASP (local search). Algorithm 25 describe the construction phase of the GRASP. The solution to be contructed is initialized in line 1 of the pseudo-code. The while loop from line 2 to 6 is repeated until the solution is fully constructed. In line 3, the restricted candidate list is built. A candidate from the list is selected, at random, in line 4 and is added to the solution in line 5. The effect of the selected solution element s on the benefits associated with every element is taken into consideration in line 6, where the greedy function is

adapted.

Algorithm 25: GRASPConstructionPhase

Input: P
$S = \{\}$
while *Solution construction not done* **do**
    $RCL \leftarrow$ build restricted candidate list
    $s \leftarrow$ select element at random from RCL
    $S = S \cup \{s\}$
    `AdaptGreedyFunction(s)`

The algorithm 26 the GRASP local search algorithm, which works in an iterative way by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The neighborhood structure N for a problem P relates a solution S of the problem to a subset of solutions $N(S)$. A solution S is said to be locally optimal if there is no better solution in $N(S)$.

Algorithm 26: GRASPLocalPhase

Input: P, $N(P)$, S
while s *not locally optimal* **do**
    find a best solution $t \in N(S)$
    $s = t$
return s

A.3.2 Evolutionary algorithms

Evolutionary algorithms (EAs) are optimization techniques that heve become popular over the last decades. EAs represent an efficient global search solution method, that can be used successfully in many highly complex applications.

The peculiar feature of an EA is that does not maintains a single current solution but a population of solutions; each solution is seen as an individual competing for survival. The idea of evolutionary algorithms, which is the same as defined by Charles Darwin, is that only the fittest individual survives within the population and it is only the fittest individual who has the higher probability of reproducing. The fittest individual is the one that has best adapted to its habitat. It works on some basic principles: the individuals evolve and compete for limited resources; the population changes dynamically, passing on only the most promising genes to future generations. This schema allows to explore the most promising areas of the space of feasible solution set.

An individual is generally represented through a certain encoding, describing the individual's chromosome, which is in turn composed of a series of genes.

The most popular type of EAs are Genetic Algorithms (GAs). Such algorithms proceed in phases, corresponding to different generational changes of the initial population. In each phase,

the following operations are applied an appropriate number of times:

- **Selection**: two (or more) solutions, named *parents*, are selected within the population, giving priority to individuals with higher fitness;
- **Crossover**: starting from the selected solutions, a certain number of *descendant* solutions are generated, by mixing the characteristics (genes) of the parent solutions;
- **Mutation**: some random modifications are applied to each generated solution, with the aim to introduce into the population new characteristics.

A genetic algorithm is characterized by a certain number of generational changes, simply called generations and coinciding with an iteration of the algorithm. GAs terminate when some termination criterion is fulfilled, i.e., after a maximum number of iterations or after a certain number of iterations in which no improve is recorded (aspiration criterion). An individual is represented by a special encoding, i.e., its chromosome, in turn the chromosome is composed of smaller parts called genes which are the items of the instance that can be used to build a solution. In each iteration to identify the strongest individuals that can be reproduced, the genetic algorithms uses a function called *fitness*, which evaluates the goodness of an individual or its ability to adapt to the habitat in which it lives, therefore the ability to survive. In many cases the *fitness* function represents the objective function of the problem under examination. Subsequently, the individuals of the current population are selected in order to generate a new population, in a pseudo-random way with a probability depending on the value of their fitness. In this regard, there are two main techniques which are:

- **Roulette wheel selection**: The fitness is used to associate a probability of selection to each individual. If f_i is the fitness of individual i in the population, its probability of being selected is (N is the number of individuals in the population):

$$p_i = \frac{f_i}{\sum_{j \in \{1,\dots,N\}} f_j}$$

  It's called roulette wheel because it can be seen as a roulette wheel in a casino, where each section of the roulette wheel is associated with an individual. Each section is larger or smaller in proportion to the individual's probability of being selected, individuals with higher probability will have larger sections and vice versa.

- **Tournament selection**: Choose few individuals at random from the population (a tournament). The individual with the best fitness (the winner) is selected for crossover.

Once the parents have been selected, through the *crossover* operation they will produce the individuals who will compose the next generation. The *crossover* operation can be seen as a function that combines the parents' genes in order to build the chromosome or chromosomes of the

children (new individuals). The new individuals in this way will inherit the best characteristics (genes) of the parents. Finally, the *mutation* operation carries out random modifications by operating on the chromosomes of the children. A mutation operation can be to reverse the value of a gene. The purpose of mutation in GAs is to introduce diversity into the sampled population. Mutation operators are used in an attempt to avoid local minima by preventing the population of chromosomes from becoming too similar to each other, thus slowing or even stopping convergence to the global optimum.

All the operations of a GA must be specialized for the problem to be solved, and one of the most important is to define the coding of a solution, this choice can influence the efficiency of the algorithm. Although for some problems in literature are used some standard implementations, for example very often is sufficient to generate an initial population in a random manner, without even guaranteeing the feasibility, allowing to obtain good solutions at the end of the execution.

A.4 Exact approaches

For the solution of Integer Linear Programming problems there not exist universally efficient methods. Very often it is necessary to use ad hoc algorithms that are able to exploit the particular structure of the problem. However, there are methods applicable to a large class of ILP problems. Among the most successful algorithms for solving ILP there is the *Branch and cut* algorithm. The *Branch and Cut* algorithm is an exact approach that has achieved great success in solving a large variety of Integer Linear programming problems. It was introduced by Padberg, Manfred and Rinaldi [PR91] and it combines two techniques: the *Branch and Bound* algorithm and the *Cutting Plane* method.

The cutting plane algorithm, uses valid inequalities in order to cut out the optimal solution of the linear relaxation, in order to restrict the formulation of the problem as much as possible, until an integer feasible solution is found. This method can be traced back to the work of Danzig, Fulkerson and Johnson [DFJ54] who used it to solve the TSP problem with 48 cities. Gomory [DFJ58] proposed the well-known fractional cuts and mixed integer cuts, a general procedure for solving mixed 0-1 problems. Branch and bound algorithm is a divide-and-conquer algorithm that iteratively divides the feasible set of a MILP problem. Implicitly lists the feasible solutions in search of a proof of optimality. This method was introduced by Land and Doig [LD10] as a general scheme for MILP problems and by Balas [Bal65] for problems 0-1. The Branch and bound algorithm was the preferred solution method in the 70s and 80s, and is now implemented in the main software libraries for solving COPs. During this period, the cutting plane algorithms were considered of less practical value as independent solution techniques.

A.4.1 Branch and Cut

A basic scheme of the branch-and-cut approach, for a minimization problem, is shown in the algorithm 27. The algorithm starts with a single subproblem to solve: the starting ILP program.

During the course of the algorithm, new subproblems are created by the branching operation: the solution of a subproblem can lead to the creation of two or more child (subproblems). This process can be represented by a tree where the nodes correspond to the subproblems and the edges represent the parent-child relationship between them. In this context, the terms node and subproblem are considered synonymous.

In algorithm 27, at line 1, a queue Q that contains the active nodes, that still need to be solved is initialized with the initial integer linear program; at line 2, the current solution (incumbent) x^* is set equal to $NULL$; at line 3 z^* which represents the current optimal value is set equal to $+\infty$. On line 4 the algorithm checks if there are still subproblems to analyze, if Q is empty on line 5 the algorithm return the optimal solution x^* with its value z^*, otherwise on line 7 extract a new subproblem P_i. At line 8 solve the linear relaxation of the subproblem P_i, named $RL(P_i)$ obtaining the solution of the subproblem x_i and its value z_i. At line 9-11, check if the $RL(Pi)$ is infeasible, in that case prune the node node P_i and go to the step 4, moving to the next subproblem. If instead the subproblem is feasible enter the if 12-28. At lines 13-15, If the value of the new solution found is not better than the incumbent one the algorithm prune P_i and go back to 4 in order to extract the next subproblem. At lines 16-20, if the value of the new solution found is better than the incumbent one and the solution x_i has only integer components, the algorithm update the incumbent solution; prune P_i and return to 4 to extract the next subproblem. Instead, if the solution found is not composed only of integer components, at lines 21-25, the algorithm applies a separation procedure to find a violated cut by x_i, in this case the algorithm returns to line 8 and solves the relaxed problem again. At lines 26-28, the algorithm, apply the branch operation to partition the subproblem P_i into two subproblems, P_j and P_t with restricted feasible regions. Add these subproblems to Q and go to Step 4.

The order in which we explore the branching tree is of great importance for the effciency of the branch-and-cut algorithm. This order is enforced by a rule for selecting the next node to be processed. It is essentially a rule for comparing two nodes. The unsolved, pending subproblems, also called active nodes, are kept in a pool. It is denoted by Q in algorithm 27. The node selection rule induces ordering of the active nodes and, as a result, the pool of active nodes can be viewed as a priority queue where the first node is the most preferred.

Algorithm 27: Branch And Cut

Input : A ILP program
Output: An optimal solution $x*$ and its objective value $z*$, or $z^* = +\infty$ if the problem is infeasible

1 $Q = \{X_{LP}\}$ #Queue of active subproblems
2 $x^* = NULL$
3 $z^* = +\infty$
4 **if** $Q = \emptyset$ **then**
5 | **return** x^* with value z^*
6 **else**
7 | $P_i \leftarrow Q.pop()$ #Select a problem P_i from the queue
8 $(x_i, z_i) \leftarrow$ solve the linear relaxation of the subproblem $RL(P_i)$
9 **if** $RL(P_i)$ *is infeasible* **then**
10 | $prune(P_i)$
11 | go to line 4
12 **if** $RL(P_i)$ *is feasible* **then**
13 | **if** $z_i \geq z^*$ **then**
14 | | $prune(P_i)$
15 | | go to line 4
16 | **if** $z_i \leq z^*$ *and* $x_i \in \mathbb{Z}^n$ **then**
17 | | $x^* = x_i$
18 | | $z^* = z_i$
19 | | $prune(P_i)$
20 | | go to line 4
21 | **if** $z_i \leq z^*$ *and* $x_i \in \mathbb{R}^n$ **then**
22 | | `SeparationProcedure`$(RL(P_i))$
23 | | **if** *cuts violated by* x_i *exists* **then**
24 | | | add cuts to $RL(P_i)$
25 | | | go to line 8
26 $P_j, P_t \leftarrow branch(P_i)$
27 $Q.push(P_j); Q.push(P_t)$
28 go to line 4

A.5 Matheuristic approaches

In the literature, the use of heuristics and metaheuristics to solve real-world problems is widespread. It is known that when modeled as optimization problems, the majority of real-world complex de-

cision problems fall into the category of NP-hard problems. This means that exact approaches, whether in business, engineering, economics or science, are bound to fail when dealing with large-scale instances. At the same time today's decision-making processes are becoming more complex and complete, as nowadays we have the need to bring optimization problems as close as possible to real-world problems, introducing new constraints and using more decision variables; at the same time more data and input parameters are available to capture the complexity of the problems themselves. The need to obtain a truthful solution requires that this must be as close as possible to the optimal one and very often it is necessary to obtain this information quickly. Matheuristics combine these two needs, retaining the computational efficiency of metaheuristics and at the same time using tools typical of exact exact methods to move towards optimality. It should be noted that the matheuristics do not guarantee the identification of the optimal solution. One of the main efforts of the scientific community is to design *general purpose* matheuristics that do not require specific knowledge of the problem and can be easily applied to a large set of classes of problems. As for metaheuristics, the strength of these paradigms is their general applicability on a large set of problems, without requiring major redesigns or any in-depth knowledge of the problem to be addressed. As a result, general paradigms seem particularly suited to obtaining a solution of the problem without investing an enormous amount of time in understanding the mathematical properties of the model and implementing a tailor-made algorithms. However, the most successful metaheuristics sometimes are also adapted to the problem or refined.

A.5.1 Kernel Search

The Kernel Search (KS) method is a matheuristic that can be used to solve MILP problems. KS was introduced by Angelelli et al. [AMGS10] in order to address the portfolio selection problem. Like any mathaeuristic, it tries to combine the strengths of exact approaches and heuristic methods. Given the set of decision variables of the MILP to be solved, the idea is to identify a subset of promising variables, i.e. variables which have a high probability of being non-zero in the optimal solution of the problem. The promising variables are inserted into a set called kernel, generally indicated with Λ, while the remaining (less promising) variables are grouped into sets called buckets. The KS requires the definition of two main parameters: *kernel_size*, the size of the kernel set, which is the set of promising variables (i.e. how many variables the kernel set must contain); *bucket_size*, the size of the buckets (the size of each individual bucket); this parameter also defines the number of buckets that must be created. The KS algorithm is divided into two phases:

- **Initialization phase**, consisting of the initial construction of the kernel and bucket sequence;
- **Extension phase**, during which variables are progressively moved from the buckets to the kernel if they improve the current solution.

The objective of the KS is to try, through a special rule, to select a set of promising variables, possibly not too large, which are representative for the problem. KS does this by using the

information provided by the optimal solution of the continuous relaxation of the input problem. The classical rule consists in sorting the in-base variables in non-increasing order of their values and sorting the out-base variables in non-increasing order their reduced cost coefficients. By giving precedence to the in-base variables, the KS populates the kernel set, taking the first $kernel\_size$ variables according to the performed sorting, and then populating the buckets, taking $bucket\_size$ variables at time. The set Λ should be small enough to allow finding, in a reasonable time, at each iteration, the optimal solution of the subproblem composed only of the variables in the kernel set plus the ones in a bucket, but also large enough to include a significant number of variables that will be part of the optimal solution. Once the initialization phase is completed, the KS solves the integer linear programming problem considering only the variables in Λ. Subsequently, the identified solution is improved through a refined search which involves the resolution of various subproblems containing the variables in Λ plus the variables contained in the i-th bucket (B_i). When a variable contained in one of the buckets is used to build a solution that is better than the previous one, this variable is moved into the kernel, so that it can be used in the next iteration together with the variables of the next bucket.