**UNIVERSITÀ
DEGLI STUDI
DI MILANO**

PhD in Computer Science, XVII cycle
Department of Computer Science "Giovanni Degli Antoni"

Optimisation and Interdiction Problems for Network Safety

INF/01

Alberto Boggio Tomasaz

Supervisor: Roberto Cordone
Coordinator: Roberto Sassi

Academic year 2023/2024

Contents

Chapter 1

Preface

The rapid evolution of science and technology is a fundamental aspect of improving collective well-being. However, it is well-known that the use of every discovery can vary greatly depending on the intentions of those who leverage such advancements. On the one hand, we are able to find increasingly effective cures for diseases that once ravaged society, improve the well-being of citizens, produce cleaner energy, and more. On the other hand, we face the danger that the same knowledge could be exploited to harm society. A striking and tragic example is the use of quantum mechanics in the Manhattan Project, which led to the creation of the first atomic bomb.

For these reasons, it is essential to study countermeasures to the potential harmful developments in science and technology, in order to ensure ethical research aimed at collective well-being. To this end, we need to develop techniques to evaluate the robustness of our systems (digital, social, infrastructural, military, and so on) and ensure effective defence and recovery strategies.

In this thesis, we will discuss optimization problems related to security. Specifically, we focus on the security of real-world systems that can be modelled using graphs or binary matrices. The first problem we consider is the *Weighted Safe Set Problem (WSSP)*, a graph optimization problem that aims to identify vertex partitions in a graph that satisfy certain dominance constraints between the parts. It is an NP-hard problem and is challenging to solve even for small instances. Next, we introduce *Binary Interdiction Problems (BIP)*, which involve two agents: the leader, called the attacker, and the follower, called the defender. The defender must solve an optimization problem after the attacker has interdicted the original instance by blocking or making certain elements more costly for the defender, with the goal of worsening the defender's optimal solution as much as possible. These are zero-sum, turn-based games whose complexity varies depending on the complexity of the defender's problem (NP, Σ_2^P, ...).

The thesis is structured as follows:

- In Chapter 2, we introduce the WSSP, providing a formal definition. We then describe some practical applications, review the scientific literature, and present some original Integer Linear Programming (ILP) models.
- In Chapter 3, we present an exact branch-and-bound algorithm for the WSSP, which leverages a continuous relaxation of the original problem, calculated using a custom algorithm.
- In Chapter 4, we describe some heuristics for the WSSP.
- In Chapter 5, we introduce Binary Interdiction Problems, particularly hard interdiction problems. We describe real-world applications and review the literature related to these problems.
- In Chapter 6, we present a new exact technique to solve these problems and some methods to enhance its efficiency.
- In Chapter 7, we present some techniques for generating resilient sets of solutions for the defender. These sets have the property that no feasible attacker's solution can simultaneously interdict all the solutions within the set.

The contents of Chapters 3 and 4 are also included in Boggio Tomasaz et al. (2023a), Boggio Tomasaz et al. (2023b), Boggio Tomasaz and Cordone (2024a) and Boggio Tomasaz and Cordone (2024b).

1.1 Implementation details

All the experiments in this thesis are conducted on a Linux server, with processor Intel Xeon E5-2620 v4 with 2.1 GHz, 16 GB of RAM and 8 cores in hyper-threading.

For what concerns the WSSP, we implemented every algorithm described in Chapters 3 and 4 in C99 and compiled the code with GNU GCC 8.3.0 with optimisation flag -O3; they run in a single thread.

For the hard interdiction part, we implemented all the algorithms in Chapters 6 and 7 in C++11 and compiled the code with `GNU G++` 8.3.0 with optimisation flag -O3. We relied on Gurobi 11.0.0 to solve the linear programming (continuous, integral or mixed) formulations implemented. This is the only multi-threading part of the code.

Chapter 2

Weighted Safe Set Problem

Given a connected undirected graph with weighted vertices, the Weighted Safe Set Problem *amounts to labelling all vertices either as safe or unsafe, in such a way that the total weight of each connected component induced by safe vertices is not exceeded by the total weight of any adjacent connected component induced by the unsafe ones. The aim is to satisfy this requirement with a minimum weight subset of safe vertices. In this chapter we describe the problem in detail and present some formulations for it.*

The *Weighted Safe Set Problem* (*WSSP*) considers a connected undirected graph $G = (V, E)$ and a weight function $w : V \to \mathbb{R}^+$ defined over its vertices.

In order to define more simply the problem, we introduce some notation:

- for any subset of vertices $C \subseteq V$, we define its weight $w(C)$ as the sum of the weights of its vertices, $w(C) = \sum_{v \in C} w_v$;
- for any two subsets of vertices $C_1, C_2 \subseteq V$, we say that they are *adjacent* and we write $C_1 \bowtie C_2$ if at least one edge $(u, v) \in E$ has $u \in C_1$ and $v \in C_2$;
- given a set of vertices $C \subseteq V$ we denote as $\mathscr{C}_G(C)$ the collection of all maximal connected components in the sub-graph of G induced by C;
- given a subset of vertices $S \subseteq V$ and its complement $U = V \setminus S$, we denote as *safety constraint* the condition that for each maximal connected component S_c in $\mathscr{C}_G(S)$ and each maximal connected component U_c in $\mathscr{C}_G(U)$ adjacent to each other, the weight of S_c is non-smaller than the weight of U_c.

$$\forall S_c \in \mathscr{C}_G(S)\ \forall U_c \in \mathscr{C}_G(U) : \ S_c \bowtie U_c \implies w(S_c) \geq w(U_c)$$

Given a non-empty subset of vertices $S \subseteq V$ and $U = V \setminus S$, if the safety constraint is satisfied for all adjacent components of the two families, we denote S as a *safe set*, the components in $\mathscr{C}_G(S)$ as *safe components* and those in $\mathscr{C}_G(U)$ as *unsafe components*. The *WSSP* searches for a safe set in G of minimum weight with respect to w.

$$S^* = \arg \min_{S \text{ safe set}} w(S) = \arg \min_{S \text{ safe set}} \sum_{v \in S} w_v$$

A visual representation can help illustrate the problem more effectively. Figure 2.1 shows a connected, undirected graph with weights (the numbers inside the vertices represent their respective weights).

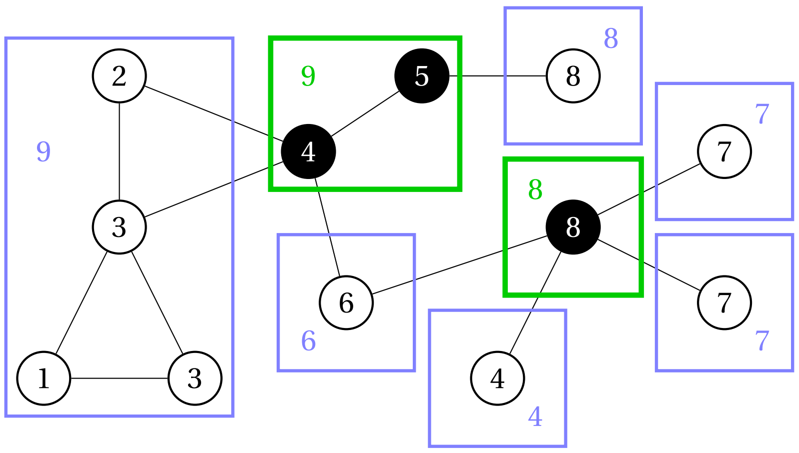

Figure 2.1: Example of WSSP instance with solution.

The solution is represented by the vertices coloured in black. These identify two safe components (in green) with a weight greater than or equal to that of each adjacent unsafe component (in blue).

In contrast, Figure 2.2 illustrates an example of an infeasible solution.

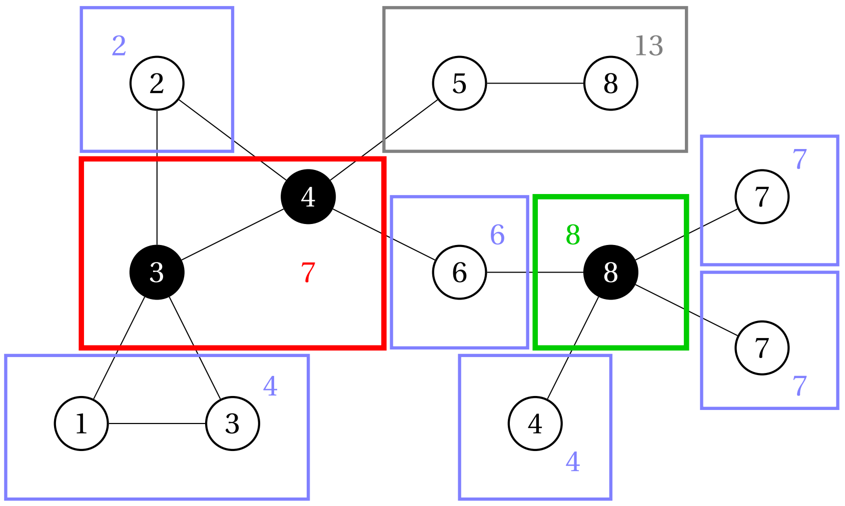

Figure 2.2: Example of violation of the WSSP constraints.

Here, the set of black vertices induces a connected component (the one in red) of weight 7 which is adjacent to a complementary connected component (the one in gray) of weight 13, violating a safety constraint.

A common variant of this problem in the literature is the *Connected Weighted Safe Set Problem (CWSSP)*, where the safe set S must induce a single safe component. In this thesis, we will not focus on this latter version, but we will occasionally mention it.

A special case of the *WSSP* is when the weight function is unitary ($\forall v \in V : w_v = 1$) and the problem is denoted as *Safe Set Problem (SSP)* and *Connected Safe Set Problem (CSSP)* when the connection requirement is imposed. In this thesis we will refer to this special case as the *unweighted* version.

The first application of the *WSSP* described in the literature is the placement of temporary refuges in buildings, to shelter people from the surrounding rooms in case of emergency, so that the area consumption is minimised (Fujita et al., 2014). The same idea can be repurposed for any graph-structured physical environment as well.

An application to social networks is the detection of a small set of users organised in communities, whose influence could allow an indirect control on all the other communities (Bapat et al., 2016). From a more general point of view, the problem can model the search for a sub-network whose control guarantees a form of preponderance over the entire network.

2.1 Literature review

The *WSSP* was first presented in Fujita et al. (2014) in the unweighted version. The same authors prove its NP-hardness and many theorems and properties about the cardinality of the optimal solution in Fujita et al. (2016). Then, Bapat et al. (2016) generalise the problem by allowing weights to assume any positive value and prove the NP-hardness of the new version even for simple graphs, like stars and trees. They also present a polynomial-time algorithm for path instances. In Fujita et al. (2016), the authors propose a linear-time algorithm for this variant on unweighted trees. The complexity of special classes of instances is studied in Àgueda et al. (2018), who show that the *WSSP* is polynomial on unweighted trees, pseudo-polynomial on bounded-treewidth graphs and interval graphs and NP-hard on unweighted planar graphs and split graphs. The parametrised complexity of the problem is investigated in Belmonte et al. (2020), proving its fixed-parameter tractability on general unweighted graphs, with a parameter depending on the size of the solution or the neighbourhood diversity. Focusing on the unweighted case and special graphs, Cordone and Franchi (2023) prove the asymptotic structure of optimal solutions for random graphs and provide upper and lower bounds of matching asymptotic size for grids.

From the computational perspective, Macambira et al. (2019) are the first to implement an exact approach, solving an *Integer Linear Programming* (*ILP*) formulation with a branch-and-cut approach to manage its exponential number of constraints. Then, Hosteins (2020) presents the first compact *Mixed-Integer Linear Programming* (*MILP*) formulation. Another branch-and-cut approach is due to Malaguti and Pedrotti (2023), who describe an *ILP* formulation with a linear number of variables.

Moving to approximation algorithms, the literature offers no contribution to approximate the optimum in polynomial-time for general graphs. In Àgueda et al. (2018) it is pointed out that, since the NP-hardness proof in Fujita et al. (2016) is based on a gap-preserving reduction from the Vertex Cover problem, the Safe Set and Connected Safe Set problems are NP-hard to approximate within a factor of 1.3606. The articles by Bapat et al. (2018) and Ehard and Rautenbach (2020) present, respectively, a polynomial-time approximation algorithm and a scheme (that is, a *PTAS*) for weighted trees. Both actually refer to the connected version of the problem, but the approximation extends to the original problem (with a worse ratio) thanks to the property that an optimal connected safe set weighs at most twice an optimal safe set.

The only heuristic approach to the *WSSP* is due to Macambira et al. (2019), who propose a polynomial-time randomized destructive heuristic, inspired by the algorithm presented in Fujita et al. (2016) for unweighted trees.

2.2 Formulations for the Weighted Safe Set Problem

As mentioned before, in the literature there are a few formulations for the WSSP. In this section we show them and provide a quick description.

Separator-based connectivity ILP formulation The branch-and-cut by Macambira et al. (2019) is the first mathematical programming approach to the WSSP and employs a MILP formulation with the following variables:

- y_i^k is a binary variable equal to 1 if vertex $i \in V$ belongs to the safe component with the smallest-index vertex $k \in V$ (i.e., k is the representative of the safe component of i); otherwise, it equals 0.
- n_i^k is a binary variable equal to 1 if vertex $i \in V$ belongs to the unsafe component with the smallest-index vertex $k \in V$ (i.e., k is the representative of the unsafe component of i); otherwise, it equals 0.
- Y_{ij}^k is a binary variable equal to 1 if vertices $i, j \in V$ belong to the safe component represented by $k \in V$; otherwise, it equals 0.
- N_{ij}^k is a binary variable equal to 1 if vertices $i, j \in V$ belong to the unsafe component represented by $k \in V$; otherwise, it equals 0.

They also define the function $\delta : \wp(V) \to \wp(V)$ ($\wp$ represents the power set function) as

$$\delta(S \subseteq V) = \{i \in V \setminus S \mid (i,j) \in E \wedge j \in S\}$$

This allows to exhibit Formulation 1

$$\min \sum_{k \in V} \sum_{i \ge k} w_i \cdot y_i^k \tag{1.1}$$

s.t.

$$\sum_{\substack{k \in V \\ k \le i}} (y_i^k + n_i^k) = 1 \qquad i \in V \tag{1.2}$$

$$y_i^k \le y_k^k \qquad i,k \in V, \quad k < i \tag{1.3}$$

$$n_i^k \le n_k^k \qquad i,k \in V, \quad k < i \tag{1.4}$$

$$\sum_{\substack{k\in V\setminus S\\ k\le i}} y_i^k + \sum_{\substack{l\in S\\ l\le j}} y_j^l \le 1 \qquad (i,j)\in E,\quad S\subset V \tag{1.5}$$

$$\sum_{\substack{k\in V\setminus S\\ k\le i}} n_i^k + \sum_{\substack{l\in S\\ l\le j}} n_j^l \le 1 \qquad (i,j)\in E,\quad S\subset V \tag{1.6}$$

$$Y_{ij}^k \le y_i^k \qquad i,j,k\in V,\quad k<i<j \tag{1.7}$$

$$Y_{ij}^k \le y_j^k \qquad i,j,k\in V,\quad k<i<j \tag{1.8}$$

$$Y_{ij}^k \ge y_i^k + y_j^k - y_k^k \qquad i,j,k\in V,\quad k<i<j \tag{1.9}$$

$$N_{ij}^k \le n_i^k \qquad i,j,k\in V,\quad k<i<j \tag{1.10}$$

$$N_{ij}^k \le n_j^k \qquad i,j,k\in V,\quad k<i<j \tag{1.11}$$

$$N_{ij}^k \ge n_i^k + n_j^k - n_k^k \qquad i,j,k\in V,\quad k<i<j \tag{1.12}$$

$$\begin{aligned}
&\sum_{\substack{u>i\\ u\ne j}} w_u\cdot y_u^i + \sum_{k<i}\left[(w_k+w_i)\cdot y_i^k + \sum_{\substack{u>k\\ u<i}} w_u\cdot Y_{ui}^k + \sum_{\substack{u>i\\ u\ne j}} w_u\cdot Y_{iu}^k\right] \ge\\
&\sum_{u\ge j} w_u\cdot n_u^j + \sum_{\substack{k<j\\ k\ne i}}\left[(w_k+w_j)\cdot n_j^k + \sum_{\substack{u>k\\ u\ne i\\ u<j}} w_u\cdot N_{uj}^k + \sum_{u>j} w_u\cdot N_{ju}^k\right] -\\
&-\sum_{k<i}\left(\sum_{\substack{u>k\\ u\ne i}} w_u\right)\cdot N_{ij}^k \qquad (i,j)\in E
\end{aligned} \tag{1.13}$$

$$\begin{aligned}
&\sum_{u\ge j} w_u\cdot y_u^j + \sum_{\substack{k<j\\ k\ne i}}\left[(w_k+w_j)\cdot y_j^k + \sum_{\substack{u>k\\ u<j\\ u\ne i}} w_u\cdot Y_{uj}^k + \sum_{u>j} w_u\cdot Y_{ju}^k\right] \ge\\
&\sum_{\substack{u>i\\ u\ne j}} w_u\cdot n_u^i + \sum_{k<i}\left[(w_k+w_i)\cdot n_i^k + \sum_{\substack{u>k\\ u<i}} w_u\cdot N_{ui}^k + \sum_{\substack{u>i\\ u\ne j}} w_u\cdot N_{iu}^k\right] -\\
&-\left(\sum_{\substack{u>i\\ u\ne j}} w_u\right)\cdot n_j^i - \sum_{k<i}\left(\sum_{\substack{u>k\\ u\ne j}} w_u\right)\cdot N_{ij}^k \qquad (i,j)\in E
\end{aligned} \tag{1.14}$$

$$\sum_{\substack{i\in\delta(S)\\ i>v_1}} y_i^{v_1} \ge y_{v_2}^{v_1} \qquad S\subset V,\ v_1=\operatorname*{arg\,min}_{i\in S} i,\quad v_2\in V\setminus(S\cup\delta(S)),\ v_2>v_1 \tag{1.15}$$

$$\sum_{\substack{i\in\delta(S)\\ i>v_1}} n_i^{v_1} \ge n_{v_2}^{v_1} \qquad S\subset V,\ v_1=\operatorname*{arg\,min}_{i\in S} i,\quad v_2\in V\setminus(S\cup\delta(S)),\ v_2>v_1 \tag{1.16}$$

$$y_j^i,\ n_j^i \in\{0,1\} \qquad i,j\in V,\ i\le j \tag{1.17}$$

$$0\le Y_{ij}^k \le 1 \qquad k,i,j\in V,\ k<i<j \tag{1.18}$$

$$0\le N_{ij}^k \le 1 \qquad k,i,j\in V,\ k<i<j \tag{1.19}$$

The objective function (1.1) represents the total weight of the safe set. The constraints (1.2) state that each vertex can only belong to one component, either safe or unsafe. The constraints (1.3, 1.4) prevent any vertex from being part of a component represented by a vertex k that is not a representative of itself. The constraints (1.5) ensure that adjacent safe vertices belong to the same safe component, and similarly, the constraints (1.6) apply to unsafe vertices. The constraints (1.7, 1.8, 1.9) assign the value 1 to the variable Y_{ij}^k only if both y_i^k and y_j^k are equal to 1. Similarly, the constraints (1.10, 1.11, 1.12) ensure that N_{ij}^k equals 1 only if both n_i^k and n_j^k are equal to 1. The constraints (1.13, 1.14) represent the safety constraints. The constraints (1.15) ensure the connectivity of vertices within the same safe component by considering vertex separators between vertices v_1 and v_2. The constraints (1.16) play a similar role to the constraints (1.15), but for unsafe vertices.

Compact flow-based MILP formulation The second approach is a branch-and-bound, due to Hosteins (2020), and exploits the first compact MILP model for the WSSP.

Let $n = |V|$ and let $w : V \to \mathbb{Q}^+$ be a weighting on the vertices. Add to G an artificial vertex 0 connected to all other vertices in the graph. The variables of the model are:

- x_i: binary variable equal to 1 if vertex i is safe; 0 otherwise.
- y_{ij}: binary variable equal to 1 if vertices i and j are adjacent and both are safe; 0 otherwise.
- y'_{ij}: binary variable equal to 1 if vertices i and j are adjacent and both are unsafe; 0 otherwise.
- a_{ic}: binary variable equal to 1 if vertex i is safe and is part of the connected component $c \in \{1, ..., n-1\}$.
- a'_{ic}: binary variable equal to 1 if vertex i is unsafe and is part of the connected component $c \in \{1, ..., n-1\}$.
- t_{ic}: binary variable equal to 1 if vertex i is safe and is the only vertex in component $c \in \{1, ..., n-1\}$ receiving non-zero flow from the artificial vertex 0.
- t'_{ic}: binary variable equal to 1 if vertex i is unsafe and is the only vertex in component $c \in \{1, ..., n-1\}$ receiving non-zero flow from the artificial vertex 0.
- f_{ij}: continuous flow variable with $i \in V \cup \{0\}$ and $j \in V$ defined for pairs of adjacent vertices (with vertex 0 being adjacent to all others).
- ω_i: total weight of the safe component to which vertex $i \in V$ belongs; 0 if i is unsafe.
- ω'_i: total weight of the unsafe component to which vertex $i \in V$ belongs; 0 if i is safe.
- v_c: total weight of the safe component $c \in \{1, ..., n-1\}$; 0 if the component is unsafe.
- v'_c: total weight of the unsafe component $c \in \{1, ..., n-1\}$; 0 if the component is safe.

Given a vertex $v \in V$, the set $N_v \subset V$ is defined as

$$N_v = \{u \in V \mid (u, v) \in E\}$$

and the quantity

$$W := \sum_{i \in V} w_i$$

$$\min \sum_{i \in V} w_i \cdot x_i \tag{2.1}$$

s.t.

$$y_{ij} \geq x_i + x_j - 1 \qquad (i,j) \in E \tag{2.2}$$
$$y'_{ij} \geq 1 - x_i - x_j \qquad (i,j) \in E \tag{2.3}$$
$$y_{ij} \leq x_i \qquad (i,j) \in E \tag{2.4}$$
$$y_{ij} \leq x_j \qquad (i,j) \in E \tag{2.5}$$
$$y'_{ij} \leq 1 - x_i \qquad (i,j) \in E \tag{2.6}$$
$$y'_{ij} \leq 1 - x_j \qquad (i,j) \in E \tag{2.7}$$
$$\sum_{j \in N_i} f_{ij} - \sum_{j \in N_i \cup \{0\}} f_{ji} = -1 \qquad i \in V \tag{2.8}$$
$$\sum_{v \in V} f_{0v} = n \tag{2.9}$$
$$f_{0i} \leq (n-1) \cdot \sum_{c=1}^{n-1} (t_{ic} + t'_{ic}) \qquad i \in V \tag{2.10}$$
$$f_{ij} \leq (n-1) \cdot (y_{ij} + y'_{ij}) \qquad (i,j) \in E \tag{2.11}$$
$$\sum_{c=1}^{n-1} a_{ic} = x_i \qquad i \in V \tag{2.12}$$
$$\sum_{c=1}^{n-1} a'_{ic} = 1 - x_i \qquad i \in V \tag{2.13}$$
$$a_{ic} \geq a_{jc} + y_{ij} - 1 \qquad (i,j) \in E, \quad c \in \{1, \ldots, n-1\} \tag{2.14}$$
$$a_{jc} \geq a_{ic} + y_{ij} - 1 \qquad (i,j) \in E, \quad c \in \{1, \ldots, n-1\} \tag{2.15}$$
$$a'_{ic} \geq a'_{jc} + y'_{ij} - 1 \qquad (i,j) \in E, \quad c \in \{1, \ldots, n-1\} \tag{2.16}$$
$$a'_{jc} \geq a'_{ic} + y'_{ij} - 1 \qquad (i,j) \in E, \quad c \in \{1, \ldots, n-1\} \tag{2.17}$$
$$t_{ic} \leq a_{ic} \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.18}$$
$$t'_{ic} \leq a'_{ic} \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.19}$$
$$\sum_{i \in V} t_{ic} \leq 1 \qquad c \in \{1, \ldots, n-1\} \tag{2.20}$$
$$\sum_{i \in V} t'_{ic} \leq 1 \qquad c \in \{1, \ldots, n-1\} \tag{2.21}$$
$$v_c = \sum_{i \in V} w_i \cdot a_{ic} \qquad c \in \{1, \ldots, n-1\} \tag{2.22}$$
$$v'_c = \sum_{i \in V} w_i \cdot a'_{ic} \qquad c \in \{1, \ldots, n-1\} \tag{2.23}$$

$$\omega_i \geq \nu_c - W \cdot (1 - a_{ic}) \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.24}$$

$$\omega_i \leq \nu_c + W \cdot (1 - a_{ic}) \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.25}$$

$$\omega_i \leq W \cdot x_i \qquad i \in V \tag{2.26}$$

$$\omega'_i \geq \nu'_c - W \cdot (1 - a'_{ic}) \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.27}$$

$$\omega'_i \leq \nu'_c + W \cdot (1 - a'_{ic}) \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.28}$$

$$\omega'_i \leq W \cdot (1 - x_i) \qquad i \in V \tag{2.29}$$

$$\omega_i \geq \omega'_j - W \cdot y'_{ij} \qquad (i,j) \in E \tag{2.30}$$

$$\sum_{i \in V} x_i \geq 1 \tag{2.31}$$

$$y_{ij}, y'_{ij} \in \{0,1\} \qquad (i,j) \in E \tag{2.32}$$

$$a_{ic}, a'_{ic} \in \{0,1\} \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.33}$$

$$\omega_i, \omega'_i \geq 0 \qquad i \in V \tag{2.34}$$

$$\nu_c, \nu'_c \geq 0 \qquad c \in \{1, \ldots, n-1\} \tag{2.35}$$

$$0 \leq f_{ij} \leq n-1 \qquad i \in V \cup \{0\}, \quad j \in V \tag{2.36}$$

$$x_i \in \{0,1\} \qquad i \in V \tag{2.37}$$

$$t_{ic}, t'_{ic} \in \{0,1\} \qquad i \in V, \quad c \in \{1, \ldots, n-1\} \tag{2.38}$$

The objective function (2.1) is the sum of all the weights of the safe vertices. The constraints (2.2–2.7) ensure the consistency of the variables y with the values taken by the variables x. These are a linearization of the constraints $y_{ij} = x_i \cdot x_j$ and $y'_{ij} = (1 - x_i) \cdot (1 - x_j)$ for each $(i,j) \in E$. The constraints (2.8) express that the incoming flow to a vertex must be equal to the outgoing flow +1 (all vertices consume one unit of flow). The constraint (2.9) imposes that the outgoing flow from the artificial vertex 0 equals n. The constraints (2.10) impose that the variables t for vertices directly receiving flow from vertex 0 take the value 1 for some component $c \in \{1, \ldots, n-1\}$. The constraints (2.11) prevent flow from circulating through edges connecting vertices from different components. The constraints (2.12, 2.13) ensure that each vertex belongs to only one safe or unsafe component. The constraints (2.14–2.17) ensure that if two vertices belong to the same component, then c, the component number, must also be the same. The constraints (2.18, 2.19) allow for a consistent numbering of the components c related to the variables a and t. The constraints (2.20, 2.21) impose that at most one vertex receives flow directly from 0 for each component. The constraints (2.22, 2.23) assign to the variables ν the sum of the weights of all the vertices that belong to the associated component. The constraints (2.24–2.29) assign to the variables ω the weight of the safe or unsafe component related to the associated vertex. The constraints (2.30) are the safety constraints. The constraint (2.31) imposes that the safe set is not empty.

Component-pair ILP formulation The third approach is another branch-and-cut, due to Malaguti and Pedrotti (2023), which uses only one binary variable for each vertex in the graph. For each vertex $v \in V$, let $x_v = 1$ if vertex v belongs to the safe set, and 0 otherwise.

Let C and C' be disjoint subsets of V. Denote by $\mathcal{P}$ the collection of such pairs (C, C') for which: $w(C) < w(C')$; $G[C]$ and $G[C']$ are connected sub-graphs; and there exists an edge $(u, v) \in E$ with $u \in C$ and $v \in C'$. Also, for any $C \subseteq V$ denote with $N(C) = \{u \in V \setminus C \mid \exists v \in C,\ (u, v) \in E\}$ the set of all external neighbours of C.

The formulation is based on the observation that for any pair $(C, C') \in \mathcal{P}$ and any safe set S of G having non-empty intersection with C, either $S \cap N(C) \neq \emptyset$ or $S \cap C' \neq \emptyset$.

$$\min \sum_{v \in V} w_v \cdot x_v \tag{3.1}$$

s.t.

$$\sum_{v \in V} x_v \geq 1 \tag{3.2}$$

$$\sum_{v \in C' \cup N(C)} x_v \geq x_l \qquad (C, C') \in \mathcal{P},\ l \in C \tag{3.3}$$

$$x_v \in \{0, 1\} \qquad v \in V \tag{3.4}$$

Inequality (3.2) enforces that at least one vertex is selected and inequalities (3.3) ensure the solution satisfies safety conditions. In fact the authors prove the following lemma (Lemma 2 in Malaguti and Pedrotti (2023)).

Lemma 1. *Let $X \subset V$ and $X \neq \emptyset$. If X is not a safe set of G, there is a pair $(C, C' \in \mathcal{P})$ such that $X \cap C \neq \emptyset$ but $X \cap C' = \emptyset$ and $X \cap N(C) = \emptyset$.*

They also provide numerous valid inequalities to strengthen formulation (3).

2.2.1 Original formulations for the Weighted Safe Set Problem

Here, we present some original formulations for which we did not conduct a comprehensive research due to poor preliminary results or lack of time.

The first formulation is not a proper ILP or MILP, but rather a combinatorial formulation that helps to understand the WSSP.

$$\min \quad w(S) = \sum_{v \in S} w_v \tag{4.1}$$

$$\text{s.t.} \quad S \subseteq V \tag{4.2}$$

$$|S| \geq 1 \tag{4.3}$$

$$w(S_i) \geq w(U_j) \qquad S_i \in \mathscr{C}_G(S), U_j \in \mathscr{C}_G(V \setminus S), S_i \bowtie U_j \tag{4.4}$$

where Constraint (4.2) states that the solution is a subset of vertices, Constraint (4.3) that it is non-empty and Constraints (4.4) impose the safety conditions.

Transitive closure formulation The next formulation is based on the concept that two vertices are in the same safe component if there is a path between them consisting entirely of safe vertices and specularly for the unsafe ones. The value L is defined as $L := \lceil \log_2(|V| - 1) \rceil$ and $M := \sum_{v \in V} w_v$. Moreover, to provide a more compact form, the logical operators $\wedge, \vee$ and $\equiv$ are implicitly formulated in the classical manner:

$$z = x \wedge y \quad \longmapsto \quad \begin{cases} z \le x \\ z \le y \\ z \ge x + y - 1 \end{cases} \qquad z = \bigwedge_i x_i \quad \longmapsto \quad \begin{cases} z \le x_i \quad \forall i \\ z \ge 1 + \sum_i (x_i - 1) \end{cases}$$

$$z = x \vee y \quad \longmapsto \quad \begin{cases} z \ge x \\ z \ge y \\ z \le x + y \end{cases} \qquad z = \bigvee_i x_i \quad \longmapsto \quad \begin{cases} z \ge x_i \quad \forall i \\ z \le \sum_i x_i \end{cases}$$

$$z = (x \equiv y) \quad \longmapsto \quad \begin{cases} z \le x - y + 1 \\ z \le y - x + 1 \\ z \ge x + y - 1 \\ z \ge 1 - x - y \end{cases}$$

$$\min \sum_{v \in V} w_v \cdot s_v \tag{5.1}$$

s.t.

$$x^0_{uv} = (s_u \equiv s_v) \qquad (u, v) \in E \tag{5.2}$$

$$x^0_{uv} = 0 \qquad (u, v) \notin E \tag{5.3}$$

$$y^l_{urv} = x^l_{ur} \wedge x^l_{rv} \qquad u, v, r \in V, \; l = 0, \ldots, L \tag{5.4}$$

$$x^{l+1}_{uv} = \bigvee_{r \in V} y^l_{urv} \qquad u, v \in V, \; l = 0, \ldots, L - 1 \tag{5.5}$$

$$\sum_{r \in V} w_r \cdot x^L_{ur} \ge \sum_{r \in V} w_r \cdot x^L_{rv} - M \cdot (1 - s_u) - M \cdot s_v \qquad (u, v) \in E \tag{5.6}$$

$$\sum_{v \in V} s_v \ge 1 \tag{5.7}$$

$$s_v \in \{0, 1\} \qquad v \in V \tag{5.8}$$

$$x^l_{uv} \in \{0, 1\} \qquad u, v \in V, \; l = 0, \ldots, L \tag{5.9}$$

$$y^l_{urv} \in \{0, 1\} \qquad u, r, v \in V, \; l = 0, \ldots, L \tag{5.10}$$

The s variables are associated to each vertex v and assume value $s_v = 1$ if v is in the safe set and 0 otherwise. Then we have the variables x^l_{uv} which assume value 1 if $s_u = s_v$ and there exists a path of length $\le 2^l$ of vertices $q \in V$ with $s_q = s_u = s_v$. Similarly, $y^l_{urv} =$

1 if and only if $s_u = s_r = s_v$ and there exist a path from u to r and another path from r to v both of length $\leq 2^l$. To ensure the correct value of x^l_{uv} we consider an inductive structure:

- The base case is when $l = 0$. If $l = 0$ it means that the variables x^0_{uv} must assume value 1 if and only if $s_u = s_v$ and u is adjacent to v (i.e., they are linked by a path of length $\leq 2^0 = 1$). Constraints (5.2) and (5.3) model exactly this case imposing that if $s_u \neq s_v$ or $(u, v) \notin E$ then $x^0_{uv} = 0$; otherwise, if $s_u = s_v$ and $(u, v) \in E$ then $x^0_{uv} = 1$.
- For the inductive step we consider that for all values $\leq l$ the hypothesis hold ($x^l_{uv} = 1$ if and only if $s_u = s_v$ and there is a path of vertices of the same kind (safe or unsafe) of length $\leq 2^l$ between them) and we prove it also for $l + 1$. From the inductive hypothesis we know that if there exist a suitable path of length $\leq 2^l$ from u to some vertex r and another such path from r to v, then $x^l_{ur} = 1 = x^l_{rv}$ and, therefore, $y^l_{urv} = 1$ from constraints (5.4) which implies $x^{l+1}_{uv} = 1$ from constraints (5.5); otherwise, if there is no suitable vertex r, it means that $x^l_{ur} = 0$ or $x^l_{rv} = 0$ implying (by constraints (5.4)) that $y^l_{urv} = 0$ for every $r \in V$. Therefore, $x^{l+1}_{uv} = 0$ by constraints (5.5).

At last, we are able to tell if two vertices lie in the same components by looking at x^L_{uv} because, since we can tell if there is a path of length $\leq 2^L$ and $L \geq \log_2(|V| - 1)$, then $2^L \geq 2^{\log_2(|V|-1)} = |V| - 1$, which is the maximum length of any path. Constraints (5.6) impose the safety constraints between the components of vertices u and v by summing the weights of all vertices in the same component. Eventually, if u is unsafe or if v is safe, the big constant M allows to satisfy the constraint automatically. At last, constraint (5.7) impose that the solution is not empty.

It is easy to prove that the integrality domain of variables x and y can be relaxed since the logical constraints preserve integrality.

Transitive closure formulation without big M A main drawback of the previous formulation is that it requires a sufficiently big coefficient M that weakens the continuous relaxation. Obviously, one should try to compute the smallest value such that the formulation still produces a feasible solution (a valid upper bound on the optimum), but we also present a slightly modified version of formulation (5) that does not require any big constant. We introduce the binary variables $p_{urv} \in \{0, 1\}$ for each edge $(u, v) \in E$ and each vertex $r \in V$. These variables are subject to the following constraints.

$$p_{uvr} = s_u \wedge (1 - s_v) \wedge x^L_{vr} \qquad (u, v) \in E, r \in V \qquad (6.1)$$

$$\sum_{r \in V} w_r \cdot x^L_{ur} \geq \sum_{r \in V} w_r \cdot p_{vur} \qquad (u, v) \in E \qquad (6.2)$$

The intuitive meaning of the variables p_{uvr} is that they assume value 1 when vertex r is in the same unsafe component as vertex v and u is safe. Therefore, we can remove constraints (5.6) from formulation (5) and insert constraints (6.1) and (6.2). Also in this case we can relax the integrality of variables p since it is guaranteed by the integrality of the variables s.

Vertex separator formulation This next formulation is similar to the previous two but, instead of propagating the connectivity, it imposes that two vertices belong to the same component if and only if there is no vertex separator of the opposite kind (unsafe for a safe component, and vice versa) that divides them.

A vertex separator $S \subseteq V$ for a connected graph G is a subset of vertices such that their removal splits the graph in two or more connected components. In our case, we are interested in (u, v)-separators where $u, v \in V$, such that the removal of the separator from the graph disconnects u and v. Another characterization is that every path from u to v (or vice versa since G is undirected) crosses at least one of the vertices in the separator. In particular, we are interested in minimal (u, v)-separators, i.e., separators that do not contain any proper subset which is itself a (u, v)-separator.

Given two non-adjacent vertices $u, v \in V$, we denote with $\mathcal{S}(u, v)$ the set of all minimal (u, v)-separators. Of course $\mathcal{S}(u, v) = \mathcal{S}(v, u)$.

The following formulation incorporates (u, v)-separators into its constraints in a way inspired by Wang et al. (2017).

$$\min \sum_{v \in V} w_v \cdot s_v \tag{7.1}$$

Subject to:

$$\sum_{v \in V} s_v \geq 1 \tag{7.2}$$

$$x_{uv} = s_u \wedge s_v \qquad (u, v) \in E \tag{7.3}$$

$$y_{uv} = (1 - s_u) \wedge (1 - s_v) \qquad (u, v) \in E \tag{7.4}$$

$$x_{uv} \geq x_{ur} + x_{rv} - 1 \qquad \{u, v, r\} \in V \tag{7.5}$$

$$y_{uv} \geq y_{ur} + y_{rv} - 1 \qquad \{u, v, r\} \in V \tag{7.6}$$

$$x_{uv} \leq \sum_{r \in S} s_r \qquad (u, v) \notin E, S \in \mathcal{S}(u, v) \tag{7.7}$$

$$y_{uv} \leq \sum_{r \in S} (1 - s_r) \qquad (u, v) \notin E, S \in \mathcal{S}(u, v) \tag{7.8}$$

$$\sum_{r \in V} w_r \cdot (x_{ur} + y_{ur}) \geq \sum_{r \in V} w_r \cdot y_{vr} \qquad (u, v) \in E \tag{7.9}$$

$$s_v \in \{0, 1\} \qquad v \in V \tag{7.10}$$

$$x_{uv} \in \{0, 1\} \qquad u, v \in V \tag{7.11}$$

$$y_{uv} \in \{0, 1\} \qquad u, v \in V \tag{7.12}$$

In this context, the variables x_{uv} behave in the same way as the variables x^L_{uv} in formulation (5), but only for the safe vertices. In fact, instead of using $L = \lceil \log_2(|V| - 1) \rceil$ "steps" to propagate the connectivity, here we force $x_{uv} = 1$ whenever there is some $r \in V$ such that $x_{ur} = 1 = x_{rv}$ with constraints (7.5). The issue lies in enforcing $x_{uv} = 0$ if u and v are in different safe components. In this regard, constraints (7.7) impose that if there exists a (u, v)-separator of unsafe vertices, then $x_{uv} = 0$. Similarly, the same considerations apply to the variables y, which represent the connectivity of the unsafe vertices. At last, for any edge $(u, v) \in E$, the safety conditions are imposed by constraints (7.9) where the right-hand-side is simply the weight of the unsafe component that contains v (0 if v is safe) whereas the left-hand-side is the weight of the component (safe or unsafe) that contains u. This way, if u, v are both unsafe and adjacent, they must be in the same unsafe component and therefore constraint (7.9) is satisfied. On the other hand, whenever v is safe, the right-hand-side evaluates to 0, satisfying the constraint. At last, when u is safe and v unsafe, the constraint enforces the safety condition.

This formulation, once again, still works if we relax the integrality constraints of variables x and y.

Obviously, the number of constraints (7.7) and (7.8) is not polynomial and, therefore, we have to resort to a branch-and-cut approach with a separation routine. The separation problem consists in the search for a pair of vertices u, v such that there exists a (u, v)-separator that violates constraints (7.7) or (7.8). In particular, if s^* is the vector containing the values of the variables s_v in some solution (of the continuous relaxation or the original problem), we can compute the most violated constraint by considering for each pair of non-adjacent vertices $u, v \in V$ the minimum (u, v)-separator with respect to s^*:

$$\mathring{S} = \min_{S \in \mathcal{S}(u,v)} \sum_{v \in S} s_v^*$$

If $\sum_{v \in \mathring{S}} s_v^* < x_{uv}^*$, where x_{uv}^* is the value of variable x_{uv} in the considered solution, then we add the associated constraint (7.7). The same can be done for constraints (7.8) considering $(1 - s_v^*)$ as weights.

Computing a minimum (u, v)-separator is a polynomial-time problem and can be done with a well-known reduction to the minimum cut problem by duplicating each vertex and connecting the pairs with an arc with capacity equal to the weight of the vertex and all other edges become directed arcs with infinite capacity. However, it is not necessary to find a minimum separator; often, a minimal separator is sufficient for our purpose.

Column generation formulation This last formulation, instead of considering binary variables s_v associated to the vertices, considers binary variables associated to connected components in the graph. In particular, let's denote with $\mathcal{C}$ the set of all (maximal or non-maximal) connected components of the graph G. If $C \in \mathcal{C}$ is a connected component, x_C is the binary variable that assumes value 1 if and only if C is a safe component in the solution and, specularly, y_C is the binary variable the assumes value 1 if and only if C is an unsafe component in the solution.

We denote with $w_C = \sum_{v \in C} w_v$ the weight of component C.

$$\min \sum_{C\in\mathcal{C}} w_C \cdot x_C \tag{8.1}$$

s.t.

$$\sum_{C\in\mathcal{C}} x_C \geq 1 \quad \left(\sum_{C\in\mathcal{C}} x_C = k\right) \tag{8.2}$$

$$\sum_{\substack{C\in\mathcal{C}:\\ u\in C}} (x_C + y_C) = 1 \qquad u \in V \tag{8.3}$$

$$\sum_{\substack{C\in\mathcal{C}:\\ u\in C\vee v\in C}} x_C \leq 1 \qquad (u,v)\in E \tag{8.4}$$

$$\sum_{\substack{D\in\mathcal{C}:\\ u\in D\vee v\in D}} y_D \leq 1 \qquad (u,v)\in E \tag{8.5}$$

$$\sum_{\substack{C\in\mathcal{C}:\\ u\in C\wedge v\notin C}} w_C \cdot x_C \geq \sum_{\substack{D\in\mathcal{C}:\\ v\in D\wedge u\notin D}} w_D \cdot y_D \qquad (u,v)\in E \tag{8.6}$$

$$x_C \in \{0,1\} \qquad C\in\mathcal{C} \tag{8.7}$$

$$y_D \in \{0,1\} \qquad D\in\mathcal{C} \tag{8.8}$$

Constraint (8.2) guarantees that the solution is non-empty. If we aim to solve the CWSSP or any other variant that imposes a given number $k \in \mathbb{N}$ of safe components, we can simply modify the constraint as showed in the parenthesis. Constraints (8.3) impose that for any vertex $u \in V$, there is exactly one safe or unsafe component that contains u. Constraints (8.4) and (8.5) make sure that if two adjacent vertices u, v are both safe or unsafe, they lie in the same component. In fact, they impose that the number of safe (or unsafe) components that contain u or v is at most 1. At last, constraints (8.6) implement the safety conditions on the edge $(u, v) \in E$, making sure that the left-hand-side only considers components $C \in \mathcal{C}$ with $u \in C$ and $v \notin C$ and the right-hand-side only considers components $D \in \mathcal{C}$ with $u \notin D$ and $v \in D$. This way, the only components considered are the ones adjacent to each other and, moreover, if both u and v are unsafe, the right-hand-side evaluates to 0 satisfying the constraint.

Of course, the number of variables in this formulation is not polynomial, therefore we should resort to column-generation to solve it. Since there are two types of variables, x and y, we require two similar pricing problems. Unfortunately, both these pricing problems are very complicated since they consist in the search for a connected component which maximises the following reduced costs:

$$\max_{C\in\mathcal{C}} \quad \mu + \sum_{u\in C} \lambda_u + \sum_{\substack{(u,v)\in E:\\ u\in C\vee v\in C}} \theta_{uv} + w_C \cdot \sum_{\substack{(u,v)\in E:\\ u\in C\wedge v\notin C}} \sigma_{uv} - w_C$$

$$\max_{D\in\mathcal{C}} \quad \sum_{u\in D} \lambda_u + \sum_{\substack{(u,v)\in E:\\ u\in D\vee v\in D}} \gamma_{uv} + w_D \cdot \sum_{\substack{(u,v)\in E:\\ u\in D\wedge v\notin D}} \sigma_{uv}$$

where μ is the dual variable associated to constraint (8.2), λ_u are the dual variables of constraints (8.3), θ_{uv} are the dual variables of constraints (8.4), γ_{uv} the ones of constraints (8.5), σ_{uv} are the dual variables of constraints (8.6) and w_C, w_D are the total weights of the component.

Disregarding the significant difficulty of this pricing problem, preliminary results show that the continuous relaxation of this formulation is of good quality.

2.3 Benchmark instances

The benchmarks considered for the WSSP in our computational experiments are available at

`https://homes.di.unimi.it/cordone/research/wssp.html`,

together with the detailed results. They can be grouped into several classes based on their topology, size and weight function. The first two classes derive from the literature.

In the following, we will indicate with $n = |V|$ the number of vertices and with $m = |E|$ the number of edges of the considered instances.

Benchmark `M` (Macambira et al., 2019) This benchmark consists of Erdős-Renyi graphs with a number of vertices covering all values from 10 to 30. For each size, three different graphs are obtained setting the number of edges to $\lfloor \delta n(n-1)/2 \rfloor$, with density $\delta \in \{0.3, 0.5, 0.7\}$. Each graph corresponds to a weighted instance, with weights randomly extracted from a uniform distribution in $\{1, \ldots, 100\}$, and an unweighted one, where all weights are unitary. Overall, therefore, this benchmark consists of $21 \cdot 3 \cdot 2 = 126$ instances. It is publicly available at

`http://www.cos.ufrj.br/~luidi/papers/safeset.html`

. The optimal values of these instances are known.

Benchmark `H` (Hosteins, 2020; Boggio Tomasaz et al., 2023a) This benchmark has a similar structure, as it also includes Erdős-Renyi graphs, but the number of vertices is $n \in \{20, 25, 30, 35, 40, 50, 60\}$, there are four classes of densities ($\delta \in \{0.1, 0.2, 0.3, 0.4\}$) and 5 instances for each size and density. Also in this case, weighted and unweighted versions of the same graphs are considered, but in the former the weights are randomly extracted from a uniform distribution in $\{1, \ldots 10\}$. Overall, this benchmark consists of $7 \cdot 4 \cdot 5 \cdot 2 = 280$ instances. The optimal values are known for most of these instances.

The new classes consist of larger instances, and will be used to investigate the performance of the algorithms, and its dependence on topological features. The number of vertices is $n \in \{100, 150, 200, 250, 300\}$ for all graphs, except for the real-world ones. As in benchmark `H`, each graph corresponds to a weighted instance, with random weights extracted from $\{1, \ldots, 10\}$, and an unweighted one.

Benchmark $\texttt{H}^+$ This is an extension of benchmark `H` with instances of bigger size. The density δ ranges in $\{0.1, 0.2, 0.3, 0.4\}$, each size and density corresponds to 5 instances. Given the five possible sizes and the two weight functions, this yields $5 \cdot 4 \cdot 5 \cdot 2 = 200$ instances.

Benchmark `GC` This consists of 9 graphs used in the 10th DIMACS challenge on Graph Clustering with a number of vertices and edges ranging from $n = 34$ and $m = 78$ to $n = 453$ and $m = 2025$. Given the two weight functions, this yields $9 \cdot 2 = 18$ instances.

Benchmark `SW` The *small-world instances* are generated with the Watts–Strogatz model (Watts and Strogatz, 1998), setting the initial degree of the vertices to $d \in \{6, 10\}$ and the randomisation parameter to $p = 0.05$. Generating 5 instances for each class yields $5 \cdot 2 \cdot 5 \cdot 2 = 100$ instances.

Benchmark `Reg` The *regular instances* have fixed degree $d \in \{5, 10\}$ with edges pairing vertices extracted at random, excluding those that already have reached the required degree. Since there are 5 graphs for each size and degree, there are $5 \cdot 2 \cdot 5 \cdot 2 = 100$ regular instances.

Benchmark `Pla` The *planar instances* are obtained distributing n points in a square with uniform random integer coordinates in $\{1, \ldots, 100\}$ and computing a greedy triangulation. As there are 5 graphs for each dimension, the planar instances are $5 \cdot 5 \cdot 2 = 50$.

Benchmark `Grid` The *grid instances* are 2D and 3D grid graphs with a toroidal structure (the vertices in the first and the last path for each dimension are linked with each other). In order to obtain the required number of vertices, we consider the following sizes for 2D graphs: 10×10, 10×15, 10×20, 10×25, 15×20, and the following ones for 3D graphs: $4 \times 5 \times 5$, $5 \times 5 \times 6$, $5 \times 5 \times 8$, $5 \times 5 \times 10$, $5 \times 6 \times 10$. Overall, there are $10 \cdot 2 = 20$ grid instances.

Overall the number of instances considered is $126 + 280 + 200 + 18 + 100 + 100 + 50 + 20 = 894$. Table 2.1 summarises their basic features, reporting for each benchmark their acronym (Name), their type and number of the instances (#), the range of the number of vertices n, the average degree $\bar{\delta}$ and the average diameter ϕ. The upper two benchmarks derive from the literature, the other ones are new.

Name	Type	#	n	$\bar{\delta}$	ϕ
`M`	random	126	[10,30]	9.5	2.5
`H`	random	240	[20,60]	8.1	4.6
`H`$^+$	random	200	[100,300]	50.1	2.5
`GC`	real-world	18	[34,453]	10.5	5.8
`SW`	smallworld	100	[100,300]	8.0	8.9
`Reg`	regular	100	[100,300]	7.5	4.7
`Pla`	planar	50	[100,300]	5.7	11.3
`Grid`	grid	20	[100,300]	5	11.1

Table 2.1: Main features of the benchmark instances.

Chapter 3

An Exact Method for the WSSP

In this chapter, we introduce a combinatorial branch-and-bound approach, whose main strength is a refined relaxation that combines graph manipulations and the solution of an auxiliary problem. We also propose fixing procedures to reduce the number of branching nodes alongside with a polynomial-time feasibility check. We present two versions of the algorithm and compare them with each other as well as with the state of the art methods from the literature, showing that our new approaches outperform all their predecessors.

Since the WSSP is a NP-hard problem, solving it to optimality requires implicit enumeration techniques like the branch-and-bound algorithm by Hosteins (2020), the branch-and-cut algorithms by Macambira et al. (2019) and Malaguti and Pedrotti (2023) or the dynamic programming approach by Àgueda et al. (2018) parametrised on the treewidth. In this chapter, we propose a combinatorial branch-and-bound for which we provide two different versions, distinguished by the subroutines they use. The algorithm proceeds by fixing vertices in or out of the solution. Each node of the branching tree, therefore, corresponds to a sub-problem in which the vertex set V is partitioned into a triplet $\langle S, U, F \rangle$, where S is the subset of *safe* vertices (fixed inside the solution), U is the subset of the *unsafe* vertices (fixed outside) and F is the subset of the residual *free* vertices. In this framework, since any feasible solution can be interpreted as a special sub-problem $(S, V \setminus S, \emptyset)$ with no free vertices, we can generalise the concepts of safe and unsafe components previously introduced by defining:

$$\begin{aligned}
&\mathscr{C}_G(S) = \{S_i \mid i = 1,2,\dots,|\mathscr{C}_G(S)|\} && \text{as the collection of all safe components } S_i,\\
&\mathscr{C}_G(U) = \{U_j \mid j = 1,2,\dots|\mathscr{C}_G(U)|\} && \text{as the collection of all unsafe components } U_j,\\
&\mathscr{C}_G(F) = \{F_\ell \mid \ell = 1,2,\dots,|\mathscr{C}_G(F)|\} && \text{as the collection of all free components } F_\ell.
\end{aligned}$$

where *safe, unsafe* and *free* components are the maximal connected components induced by S, U and F, respectively. Moreover, in this chapter, we will denote the number of safe components in $\mathscr{C}_G(S)$ as

$$k := |\mathscr{C}_G(S)|$$

In each node of the branching tree, the algorithm tries to construct a feasible solution, respecting the current assignments. It computes a lower bound on the optimum of the sub-problem. It fixes some free vertices using logical tests. Finally, it decides how to branch creating two children nodes.

The following proposition provides a technical remark that will be used by the upper and lower bounding procedures. The intuitive idea is that, when the free vertices of a sub-problem $\langle S, U, F\rangle$ are assigned to the safe and the unsafe components to generate a solution, each original component can be enlarged and possibly merged with other components of the same type; therefore, each safe component of the sub-problem is fully included in exactly one safe component of the solution, and each unsafe component of the sub-problem is fully included in exactly one of the unsafe components of the solution.

Proposition 1. *Let $\langle S, U, F\rangle$ be a sub-problem and $\tilde{S}$ a set of vertices such that $S \subseteq \tilde{S} \subseteq S \cup F$, that is a feasible or infeasible solution for the sub-problem. For each component $S_i \in \mathscr{C}_G(S)$ there is a component $\tilde{S}_x \in \mathscr{C}_G(\tilde{S})$ such that $S_i \subseteq \tilde{S}_x$ and $w(S_i) \leq w(\tilde{S}_x)$; for each component $U_j \in \mathscr{C}_G(U)$ there is a component $\tilde{U}_y \in \mathscr{C}_G(V \setminus \tilde{S})$ such that $U_j \subseteq \tilde{U}_y$ and $w(U_j) \leq w(\tilde{U}_y)$.*

Proof. Consider a safe component $S_i \in \mathscr{C}_G(S)$. Since S_i is connected, we know that every pair of vertices $u, v \in S_i$ are connected through a path involving only vertices of S_i. Since $S \subseteq \tilde{S}$, all the vertices in the path between u and v are also contained in $\tilde{S}$, but this implies that they are contained in the same component $\tilde{S}_x \in \mathscr{C}_G(\tilde{S})$. This holds for each pair of vertices $u, v \in S_i$, implying that S_i is fully contained in $\tilde{S}_x$. As the weights are non-negative, $S_i \subseteq \tilde{S}_x$ implies $w(S_i) \leq w(\tilde{S}_x)$. The same reasoning can be applied to the unsafe components, as $U \subseteq V \setminus \tilde{S} \subseteq U \cup F$. □

3.1 Upper bound and feasibility

Once the safe, unsafe and free components of a sub-problem are known, it is possible either to find a feasible solution, or to prove that none exists. In particular, the definition below identifies free components whose vertices belong to the unsafe set in any feasible solution.

Definition 1. *A free component $F_\ell \in \mathscr{C}_G(F)$ is called **unsavable** if:*

1. *F_ℓ is not adjacent to any safe component;*
2. *its weight $w(F_\ell)$ is smaller than the weight of at least one adjacent (unsafe) component.*

The following proposition shows that all vertices of the unsavable free components can be moved from subset F to U, with no risk of excluding feasible solutions.

Proposition 2. *In all feasible solutions of sub-problem $\langle S, U, F\rangle$, the vertices of the unsavable free components are unsafe.*

Proof. Let $F_\ell \in \mathscr{C}_G(F)$ be an unsavable free component, and $U_j \in \mathscr{C}_G(U)$ an unsafe component adjacent to F_ℓ such that $w(U_j) > w(F_\ell)$. Let $\tilde{S}$ be a feasible solution and

$\tilde{U}_j \in \mathscr{C}_G(V \setminus \tilde{S})$ the unsafe component that contains U_j (Proposition 1). Either $\tilde{U}_j$ fully includes also F_ℓ, or there exists a vertex $v \in F_\ell$ that is adjacent to $\tilde{U}_j$ and belongs to $\tilde{S}$. The first case implies the thesis. The second case contradicts the feasibility of $\tilde{S}$. In fact, let $\tilde{S}^{(v)} \in \mathscr{C}_G(\tilde{S})$ be the safe component that includes v in $\tilde{S}$. Since F_ℓ is unsavable, $\tilde{S}^{(v)} \subseteq F_\ell$, but all weights are non-negative, and therefore $w(\tilde{S}^{(v)}) \leq w(F_\ell) < w(U_j) \leq w(\tilde{U}_j)$, which violates the safety constraint. □

Since moving free vertices to U enlarges some unsafe components, other free components can become unsavable, allowing to apply the proposition repeatedly. At the end of this chain effect, moving all free vertices to the safe set and checking the safety requirements on the resulting set provides a simple feasibility test, as proved in the following theorem.

Theorem 1. *A sub-problem $\langle S, U, F\rangle$ with no unsavable free component is feasible if and only if $S \cup F$ is a feasible solution.*

Proof. If $S \cup F$ is a feasible solution, the sub-problem is trivially feasible. If, on the contrary, $S \cup F$ is infeasible, there is a maximal connected component $\hat{S}_i \in \mathscr{C}_G(S \cup F)$ that is adjacent to a maximal connected component $U_j \in \mathscr{C}_G(U)$, and has strictly lower weight: $w(\hat{S}_i) < w(U_j)$. Consider any $\tilde{S} \subseteq S \cup F$. According to Proposition 1, component U_j is fully contained in a component $\tilde{U}_y \in \mathscr{C}_G(V \setminus \tilde{S})$. On the other hand, when $S \cup F$ is reduced to $\tilde{S}$ moving vertices from F to U, component $\hat{S}_i$ can become smaller, or split in smaller disjoint subsets, or completely vanish. The third case is impossible, because it would require $\hat{S}_i$ to fully consist of free vertices and have no adjacent safe vertex: such a component, with $w(\hat{S}_i) < w(U_j)$, would be unsavable, against the hypothesis. Therefore, $\tilde{S}$ contains one or more components $\tilde{S}_x \subseteq \hat{S}_i$, with weights $w(\tilde{S}_x) \leq w(\hat{S}_i) < w(U_j) \leq w(\tilde{U}_y)$ and one of these components is adjacent to $\tilde{U}_y$. Consequently, $\tilde{S}$ is infeasible. □

The solution provided by the previous theorem for feasible sub-problems is actually the worst feasible one, but it is a fast way to potentially improve the upper bound during the visit of the branching tree.

3.2 Lower bound

The lower bounding procedure adopted by the algorithm exploits a sequence of relaxations, obtained by changing the topology of the graph and removing or weakening the safety constraints, until we obtain a much simpler problem, for which we build an *ILP* formulation, whose continuous relaxation we solve via an *ad hoc* algorithm.

At each node of the branching tree, it is possible to compute two simple lower bounds, based on the weights of the safe and unsafe components of sub-problem $\langle S, U, F\rangle$. The first bound is trivially the total weight of the safe vertices: for each feasible solution $\tilde{S}$, in fact, $w(\tilde{S}) \geq w(S)$. The second bound is based on the safety constraints: in any feasible solution, the unsafe component of maximum weight must be dominated by the adjacent safe components, and consequently by the total weight of the solution. Hence, the maximum weight of any connected component of $\mathscr{C}_G(U)$ provides a lower bound on the optimum: $w(\tilde{S}) \geq \max_{U_j \in \mathscr{C}_G(U)} w(U_j)$.

Since these bounds do not take into account the weights of the free vertices, however, their quality in the upper levels of the branching tree is usually poor. The development of a more refined bound is the main contribution of this chapter and the main reason for the effectiveness of the branch-and-bound algorithm. This bound is based on the remark that all free vertices that are adjacent to both S and U will be included in the existing safe and unsafe components, without generating any new one. Therefore, these free vertices can be used to tighten the previous two bounds. As an example, Figure 3.1 shows a sub-problem $\langle S, U, F\rangle$ in which $S = \{1,6\}$, $U = \{3,7\}$ and $F = \{2,4,5\}$. The

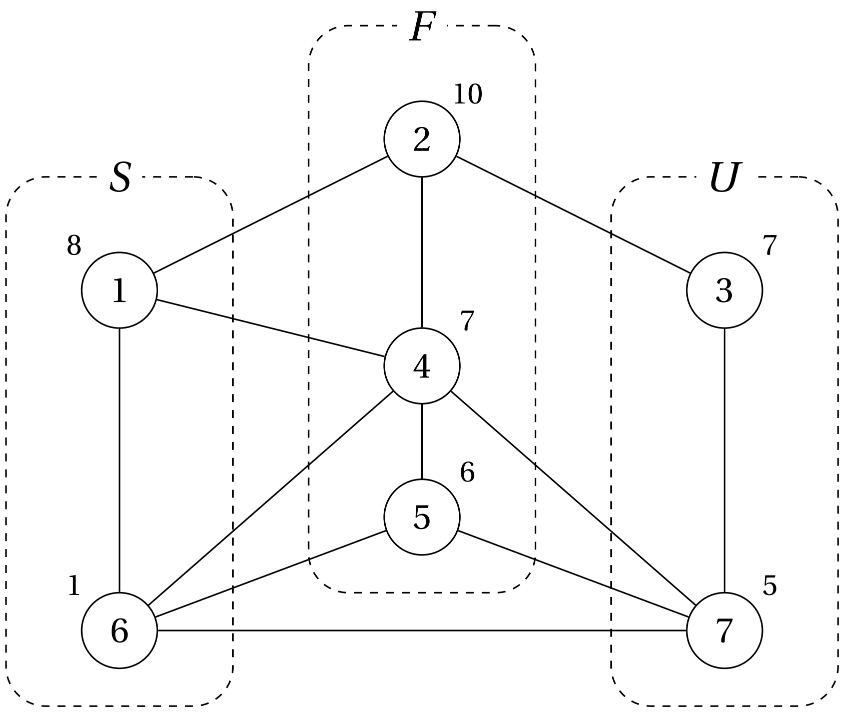

Figure 3.1: Example of a sub-problem $\langle S, U, F\rangle$. The weight of each node is displayed next to it.

two simple lower bounds are $w(S) = 9$ and $w(U) = 12$. However, all the vertices in F are adjacent to both S and U and, therefore, in every feasible solution their total weight ($w(F) = 23$) will be distributed between S and U. The distribution that minimises the maximum weight of the two components corresponds to first increasing $w(S)$ by 3 to level them off and then dividing the residual weight of F in equal parts between S and U, thus obtaining a lower bound equal to $9+3+(23-3)/2 = 22$. Incidentally, this is also the value of the optimal solution of the sub-problem, $S^* = \{1,4,5,6\}$.

Such a situation is quite frequent, especially when the graph is dense, as shown by the following remark based on the model proposed by Gilbert (1959).

Remark 1. *Let $G = (V, E)$ be a random graph following the Erdős-Rényi-Gilbert model, where each pair of vertices has a fixed probability $\delta \in [0,1]$ of corresponding to an edge.*

Given a sub-problem $\langle S, U, F\rangle$, the expected cardinality of the set of free vertices adjacent to both S and U, $X = \{x \in F \mid \exists s \in S,\ u \in U:\ (s,x) \in E\ \wedge\ (x,u) \in E\}$, is

$$\mathbb{E}[|X|] = |F| \cdot (1 - (1-\delta^2)^{|S|\cdot|U|}).$$

Proof. The probability that some vertex $f \in F$ belongs to X is

$$\begin{aligned}
\mathbb{P}(f \in X) &= \mathbb{P}(\exists s \in S, u \in U : (s,f) \in E \wedge (f,u) \in E) \\
&= 1 - \mathbb{P}(\forall s \in S, u \in U : (s,f) \notin E \vee (f,u) \notin E) \\
&= 1 - \prod_{s\in S}\prod_{u\in U} \mathbb{P}((s,f) \notin E \vee (f,u) \notin E) \\
&= 1 - \prod_{s\in S}\prod_{u\in U} (1 - \mathbb{P}((s,f) \in E) \cdot \mathbb{P}((f,u) \in E)) \\
&= 1 - \prod_{s\in S}\prod_{u\in U} (1 - \delta \cdot \delta) \\
&= 1 - (1-\delta^2)^{|S|\cdot|U|}
\end{aligned}$$

Let X_f be the binary random variable equal to 1 when $f \in X$, and 0 otherwise. The cardinality of X is the random variable $|X| = \sum_{f\in F} X_f$, and therefore, a binomial variable with parameters $|F|$ and $\mathbb{P}(f \in X)$. Its expected value is $\mathbb{E}[|X|] = |F| \cdot \mathbb{P}(f \in X)$, which implies the thesis. □

The example of Figure 3.1 conveys the general idea, but is simplified from several points of view. First of all, the number of safe or unsafe components can be larger than one, and adding free vertices can merge components, instead of simply enlarging them. Then, not all free vertices are adjacent both to the safe and unsafe components. Also, the assignment of free vertices to the components with the aim to reduce the maximum weight bears a resemblance to the subset sum problem, but is also constrained by the topology of the graph. Finally, such an NP-complete problem is hard to solve.

In order to deal with all these aspects, we propose two versions of the refined lower bound. Both versions consider the weights of the free vertices adjacent to both S and U and the computation is based on the idea of making a conservative guess of how the weights of some such vertices will redistribute between S and U. The first (*strict lower bound*) is theoretically stronger but requires stronger assumptions which might reduce effectiveness of other components of the algorithm. The second (*flexible lower bound*) is theoretically weaker but does not require any assumption.

3.2.1 Strict lower bound

In this version, we assume that:

1. S and U are not empty;
2. at most one of the $|\mathscr{C}_G(S)| = k \geq 1$ components of $\mathscr{C}_G(S)$ is adjacent to F.

The first limitation is true in most sub-problems, and the second limitation can be easily maintained in all nodes by adopting a branching rule described in Section 3.4.

Under this assumption, no pair of such components will be merged by fixing free vertices in the solution. Without any loss of generality, we can impose that, for any feasible solution $\tilde{S}$, the first safe components are the same as in S ($\tilde{S}_i = S_i$ for $i = 1, \dots, k-1$), S_k possibly incorporates free vertices ($\tilde{S}_k \supseteq S_k$), and new safe components $\tilde{S}_i$ ($i = k+1, \dots, r$) are possibly created from scratch. Figure 3.2 provides examples of the possible cases.

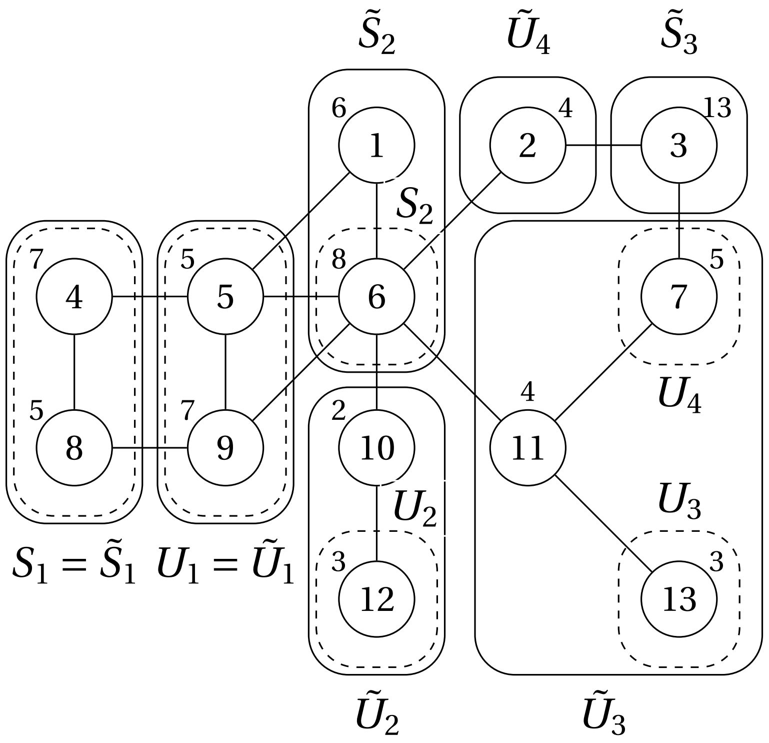

Figure 3.2: The safe and unsafe components of sub-problem $\langle S, U, F \rangle$ (marked in dashed lines) satisfy two basic limitations: S and U are non-empty, and at most one of the safe components of $\mathscr{C}_G(S)$, namely S_2, is adjacent to F. Considering any feasible solutions of the sub-problem (e.g, $\tilde{S} = \{1,3,4,6,8\}$), its safe and unsafe components (marked in continuous lines) can remain the same ($\tilde{S}_1 = S_1$ and $\tilde{U}_1 = U_1$), be augmented ($\tilde{S}_2 = S_2 \cup \{1\}$ and $\tilde{U}_2 = U_2 \cup \{10\}$) or be created from scratch ($\tilde{S}_3 = \{3\}$ and $\tilde{U}_4 = \{2\}$); only the unsafe components can merge including free vertices ($\tilde{U}_3 = U_3 \cup \{11\} \cup U_4$).

Given sub-problem $\langle S, U, F \rangle$ with $S = \{4,6,8\}$ and $U = \{5,7,9,12,13\}$, set S consists of $k = 2$ components: $S_1 = \{4,8\}$ cannot be augmented because it is not adjacent to $F = \{1,2,3,10,11\}$, whereas $S_2 = \{6\}$ can. In fact, the feasible solution $\tilde{S} = \{1,3,4,6,8\}$ induces three safe components: the first one is unchanged with respect to S ($\tilde{S}_1 = S_1$), the second one is augmented ($\tilde{S}_2 = \{1,6\} \supset S_2$) and the third one is completely new ($\tilde{S}_3 = \{3\}$). Considering the unsafe components, set U induced four: $U_1 = \{5,9\}$, $U_2 = \{12\}$, $U_3 = \{13\}$ and $U_4 = \{7\}$. The complement of the feasible solution $\tilde{U} = V \setminus \tilde{S} = \{2,5,7,9,10,11,12,13\}$ has the following unsafe components: $\tilde{U}_1$ coincides with U_1, $\tilde{U}_2$ is obtained augmenting U_2 by vertex 10, $\tilde{U}_3$ is obtained merging U_3 and U_4

through vertex 11, and $\tilde{U}_4$ is created from scratch including vertex 2. This confirms the properties stated by Proposition 1. Additionally, the limitation on sub-problem $\langle S, U, F\rangle$ guarantees that none of the safe components of S merge.

Now, we focus on the subset of free vertices that are adjacent to both S and U:

$$F' = \{v \in F \mid \exists s \in S, u \in U : (s, v) \in E \wedge (v, u) \in E\}.$$

Under the limitation above introduced, the vertices of F' are actually adjacent only to safe component S_k. In any feasible solution, these vertices either join S_k, increasing its weight, or join an unsafe component U_ℓ, increasing its weight and possibly merging it with other unsafe components. Since the second situation is harder to treat, we avoid it modifying the graph so that also the existing unsafe components cannot merge. The following propositions show that this can be done while simultaneously producing a relaxation of the original sub-problem. In fact, the manipulations described limit the ways in which the unsafe components U_ℓ can enlarge, but they keep and possibly extend those in which the safe component S_k can become larger.

Proposition 3. *Removing an edge (u, v) from graph $G = (V, E)$ provides a relaxation of sub-problem $\langle S, U, F\rangle$ if $u \in U$ and $v \in F$ or both $u, v \in F'$.*

Proof. We now show that any feasible solution $\tilde{S}$ remains feasible in the graph obtained removing edges as in the statement, and therefore the resulting problem is a relaxation of the original one.

Let (u, v) be an edge with $u \in U$ and $v \in F$. If $v \in \tilde{S}$, the safe and unsafe components of $\tilde{S}$ that include, respectively, v and u are adjacent in G, but can be non-adjacent in the reduced graph. This would remove a safety constraint. If on the contrary $v \in V \setminus \tilde{S}$, u and v belong to the same unsafe component, and removing the edge could split it, replacing some safety constraints with other ones concerning unsafe components of lower weight. In both cases, the feasibility of $\tilde{S}$ is maintained.

Let (u, v) be an edge with $u, v \in F'$. If one of the two vertices is in $\tilde{S}$, say u, and the other in $V \setminus \tilde{S}$, say v, deleting the edge does not cause any change because $u \in \tilde{S}_k$ and v is directly adjacent also to $\tilde{S}_k$. If both are unsafe, deleting the edge can split the unsafe component that includes them, replacing some safety constraints with weaker ones. Finally, if both u and v are safe, since they are adjacent to S_k and S_k is connected, they both belong to component $\tilde{S}_k$, that remains connected. Hence, $\tilde{S}$ remains feasible. □

A technical detail worth discussing is that removing edges can disconnect the graph, while the *WSSP* is commonly defined on connected graphs. With the purpose to obtain a problem for which lower bounds are simple to compute, we adopt the natural extension to non-connected graphs that admits the existence of isolated unsafe components, because they do not violate any safety constraint.

Proposition 4. *Replacing in graph $G = (V, E)$ an edge (u, v) with $u \in F'$ and $v \in F \setminus F'$ with any edge (s, v) with $s \in S$ provides a relaxation of sub-problem $\langle S, U, F \rangle$.*

Proof. Consider any feasible solution $\tilde{S}$ and let (u, v) be an edge with $u \in F'$ and $v \in F \setminus F'$. We add the edge (s, v) with $s \in S$ and denote with $S_x \in \mathscr{C}_G(S)$ the safe component that includes s. In the context of the *strict lower bound*, we require that $S_x = S_k$, for the *flexible lower bound* it can be any safe component. If $v \in \tilde{S}$, the safe component of $\tilde{S}$ that includes v merges with S_x (unless they already were the same), the weight of their union is larger than the original weights and all safety constraints are even more strongly satisfied. If $v \in V \setminus \tilde{S}$ and $u \in \tilde{S}$ the adjacent components induce the same safety constraint. Finally, if $u, v \in V \setminus \tilde{S}$, the unsafe component including u and v possibly splits, inducing weaker safety constraints. □

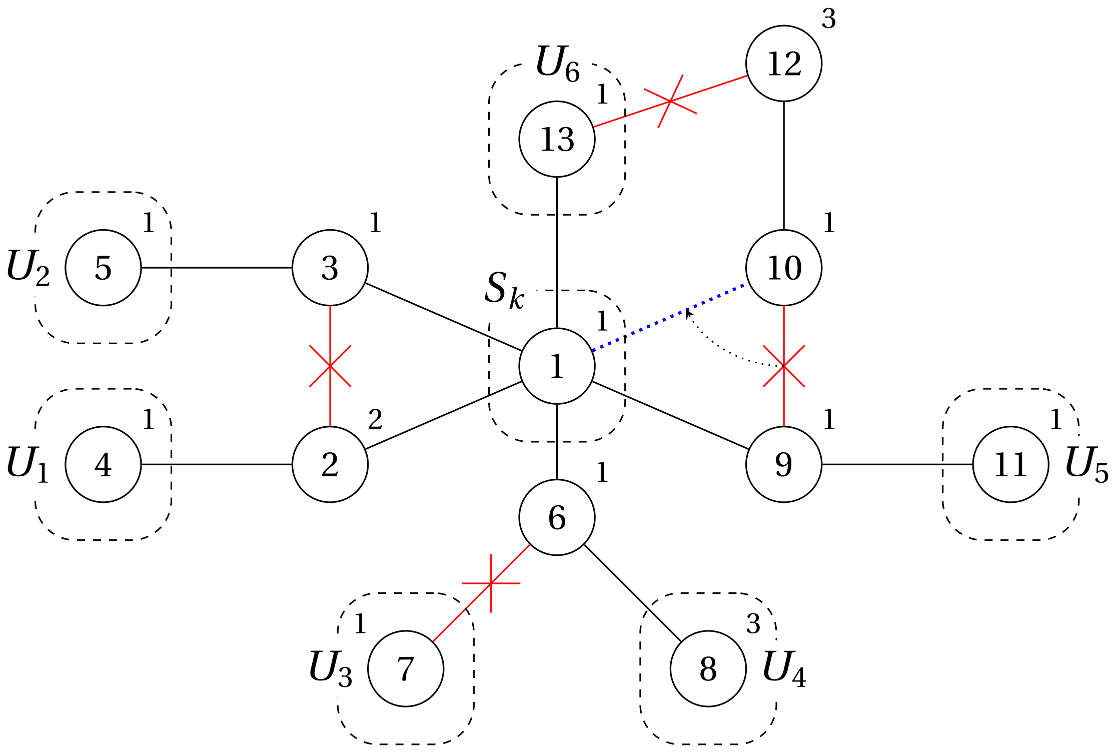

Figure 3.3: Example of graph manipulations: removing each of the three edges (2,3), (6,7) or (12,13) (in red) or replacing edge (9,10) (also in red) with (1,10) (in dotted blue) yields a relaxation of the original problem.

Figure 3.3 shows an example of these relaxations for graph $G = (V, E)$ with $V = \{1, \ldots, 13\}$ and the sub-problem with $S = \{1\}$, $U = \{4, 5, 7, 8, 11, 13\}$ and $F = \{2, 3, 6, 9, 10, 12\}$). The dashed boxes are the only safe component $S_k = \{1\}$ and the six unsafe components $U_1 = \{4\}$, $U_2 = \{5\}$, $U_3 = \{7\}$, $U_4 = \{8\}$, $U_5 = \{11\}$, $U_6 = \{13\}$, while $F' = \{2, 3, 6, 9\}$. The weights are reported next to each vertex.

Table 3.1 provides examples of all the modifications discussed in the previous propositions on the graph of Figure 3.3. The removed or replaced edges are marked with crosses, the new edge is drawn with a dotted line and an arrow points it from the original edge. The first row of the table describes the starting graph, the optimal solution S^* and its total weight $w(S^*)$. Each of the following rows reports a modification applied to the starting graph, the corresponding proposition, the optimal solution S^* of the relaxed instance obtained and its total weight $w(S^*)$. The last row applies all the previous modifications. In all cases, the optimal solution of the original problem remains feasible, but a new better solution appears; in particular the value of the last one coincides with the simple bound $w(U_4) = 3$.

Modification	Proposition	S^*	$w(S^*)$
nothing		$\{1,3,6,9,10\}$	5
remove $(2,3)$	3	$\{1,6,9,10\}$	4
remove $(6,7)$	3	$\{1,3,9,10\}$	4
remove $(12,13)$	3	$\{1,3,6,9\}$	4
replace $(9,10)$ with $(1,10)$	4	$\{1,3,6,10\}$	4
all the previous	3, 4	$\{1,6,10\}$	3

Table 3.1: Manipulations applied to the graph in Figure 3.3 and their effects on the optimal solution.

The lower bounding procedure adopted applies Propositions 3 and 4 to graph G so as to obtain a reduced graph G' in the following way:

- remove all edges (u,v) with $u, v \in F'$;
- remove all edges (u,v) with $u \in U$ and $v \in F \setminus F'$;
- for each vertex $v \in F'$, remove all but one of the edges (u,v) such that $u \in U$, so that at most one unsafe component remains adjacent to v;
- replace every edge (u,v) with $u \in F'$ and $v \in F \setminus F'$ with an edge (s,v) with $s \in S_k$ (if such an edge already exists, just remove (u,v)).

Thanks to these modifications, in the reduced graph the unsafe components $U_\ell \in \mathscr{C}_G(U)$ can include vertices of F', but not merge with each other, and they cannot extend to $F \setminus F'$. Therefore, in any feasible solution $\tilde{S}$ of the relaxed problem each unsafe component includes at most one unsafe component U_ℓ and can be denoted by the same index $\tilde{U}_\ell \in \mathscr{C}_G(V \setminus \tilde{S})$. Finally, each $v \in F'$ joins either S_k or the only unsafe component U_ℓ adjacent to v.

The choice of the edge between each $v \in F'$ and U is arbitrary, but, considering that the maximum weight of an unsafe component provides a lower bound on the optimum, it looks more promising to allow a vertex to join an unsafe component of large weight, rather than a component of small weight. Therefore, we associate each vertex to the adjacent unsafe component of maximum weight, obtaining a partition of F' into subsets associated to the unsafe components U_ℓ.

$$F'_\ell = \left\{ v \in F' \mid U_\ell = \arg \max_{\substack{U_y \in \mathscr{C}_G(U): \\ U_y \bowtie_G \{v\}}} w(U_y) \right\} \qquad \ell \in L',$$

where $L' \subseteq \{1,\dots,|\mathscr{C}_G(U)|\}$ is the set of indices of the unsafe components U_ℓ that are adjacent to S_k or to F', and consequently of the unsafe components $\tilde{U}_\ell$ that can be adjacent to and impose safety constraints on $\tilde{S}_k$ in the original graph. Notice that some subsets F'_ℓ can be empty, either because the corresponding unsafe component U_ℓ is not adjacent to F' or because it is of small weight. For example, after applying all the manipulations listed in Table 3.1 to Figure 3.3, F' is partitioned into $F'_1 = \{2\}$, $F'_2 = \{3\}$, $F'_3 = \emptyset$, $F'_4 = \{6\}$, $F'_5 = \{9\}$, $F'_6 = \emptyset$.

Now, we can go back to the combinatorial definition of the *WSSP* . Any feasible solution $\tilde{S}$ of sub-problem $\langle S, U, F\rangle$ on graph G' must satisfy the safety constraints

$$w(\tilde{S}_i) \geq w(\tilde{U}_\ell) \qquad \tilde{S}_i \in \mathscr{C}_{G'}(\tilde{S}), \tilde{U}_\ell \in \mathscr{C}_{G'}(V \setminus \tilde{S}) : \tilde{S}_i \bowtie_{G'} \tilde{U}_\ell, \tag{9}$$

We relax all of them, except for those that involve $\tilde{S}_k$ and $\tilde{U}_\ell$ with $\ell \in L'$, that we rewrite as

$$w(\tilde{S}_k) = w(S_k) + \sum_{v \in F \cap \tilde{S}_k} w_v x_v \geq w(U_\ell) + \sum_{v \in F'_\ell} w_v(1-x_v) = w(\tilde{U}_\ell) \quad \ell \in L', \tag{10}$$

where the binary variables $x_v \in \{0,1\}$ for $v \in F$ are set to 1 if vertex $v \in \tilde{S}$, to 0 otherwise. The left-hand-side of the inequality can be majorised (relaxing the constraint) by simply replacing $F \cap \tilde{S}_k$ with F. Intuitively, this corresponds to allowing free vertices that are not in component $\tilde{S}_k$ to contribute to its safety. The advantage is to remove any term depending on $\tilde{S}$ (which is unknown *a priori*), apart from the binary variables x_v. The resulting relaxation can be formulated as:

$$\min w(S) + \sum_{v \in F} w_v x_v \tag{11.1}$$

$$w(S_k) + \sum_{v \in F} w_v x_v \geq w(U_\ell) + \sum_{v \in F'_\ell} w_v(1-x_v) \qquad \ell \in L' \tag{11.2}$$

$$x_v \in \{0,1\} \qquad v \in F. \tag{11.3}$$

This problem generalises the optimisation version of the *Subset Sum problem*, that is NP-complete (Garey and Johnson, 1979): setting $F' = F$, $L' = \{1\}$ and $w(S_k) = w(U_1)$ yields $\min \sum_{v\in F} w_v x_v$ subject to $\sum_{v \in F} w_v x_v \geq 1/2 \cdot \sum_{v\in F} w_v$. Since we need to solve problem (11) at every branching node, we consider its continuous relaxation. Let us introduce the auxiliary variables $\sigma = \sum_{v\in F} w_v x_v$ and $\tau_\ell = \sum_{v \in F'_\ell} w_v(1-x_v)$ with $\ell \in L'$. These variables satisfy the following properties:

$$\tau_\ell = \sum_{v\in F'_\ell} w_v(1-x_v) \leq \sum_{v \in F'_\ell} w_v = w(F'_\ell) \qquad \ell \in L' \tag{12}$$

$$\begin{aligned}\sigma + \sum_{\ell \in L'} \tau_\ell &= \sum_{v \in F \setminus F'} w_v x_v + \sum_{v\in F'} w_v x_v + \sum_{\ell \in L'} \sum_{v \in F'_\ell} w_v(1-x_v) \\ &= \sum_{v \in F\setminus F'} w_v x_v + \sum_{v \in F'} w_v x_v + \sum_{v\in F'} w_v(1-x_v) \\ &\geq w(F')\end{aligned} \tag{13}$$

from which we derive

$$\begin{aligned}
&\min \sigma && && (14.1)\\
&w(S_k) + \sigma \geq w(U_\ell) + \tau_\ell && \ell \in L' && (14.2)\\
&\tau_\ell \leq w(F'_\ell) && \ell \in L' && (14.3)\\
&\sigma + \sum_{\ell \in L'} \tau_\ell \geq w(F') && && (14.4)\\
&\sigma, \tau_\ell \geq 0 && \ell \in L' && (14.5)
\end{aligned}$$

Notice that constraint (14.4) could be considered in equality version, for some instances, thanks to the following remark.

Remark 2. *Given a feasible solution σ, τ_ℓ of formulation* (14) *with $\sigma + \sum_{\ell \in L'} \tau_\ell > w(F')$, if $\sigma \leq w(F')$ then there exists an equivalent (same objective value) feasible solution σ, τ'_ℓ such that $\sigma + \sum_{\ell \in L'} \tau'_\ell = w(F')$.*

Proof. To obtain such solution we just execute the following procedure:

1. initialize $\tau'_\ell = \tau_\ell$ for every $\ell \in L'$;
2. select the index $\bar{\ell} = \arg\max_{\ell \in L'} \tau'_\ell$;
3. compute $\Omega = \sigma + \sum_{\ell \in L'} \tau'_\ell - w(F')$ as the surplus of constraint (14.4);
4. if $\tau'_{\bar{\ell}} > \Omega$, then decrease $\tau'_{\bar{\ell}}$ by Ω and stop;
5. otherwise, set $\tau'_{\bar{\ell}} = 0$ and proceed with step 2.

The procedure ends because at most $|L'|$ variables are set to 0. In fact, if all variables $\tau'_\ell = 0$, then $\sigma + \sum_{\ell \in L'} \tau'_\ell = \sigma + 0 \leq w(F')$ meeting the stopping criteria. Moreover, at the end of the procedure, the last selected variable $\tau'_{\bar{\ell}}$ is decreased by Ω meaning that

$$\sigma + \sum_{\ell \in L'} \tau'_\ell - \Omega = \sigma + \sum_{\ell \in L'} \tau'_\ell - \left(\sigma + \sum_{\ell \in L'} \tau'_\ell - w(F') \right) = w(F')$$

□

To solve problem (14) and compute a lower bound for sub-problem $\langle S, U, F \rangle$, we propose Algorithm 1.

First, by setting $\tau_\ell = 0$ for all $\ell \in L'$ and $\sigma = \max\{0, \max_{\ell \in L'} w(U_\ell) - w(S_k)\}$, we obtain the best solution that satisfies all constraints but (14.4). Then, we increase each τ_ℓ variable to reduce the infeasibility of (14.4), without violating constraints (14.2, 14.3). The main loop of the algorithm iteratively computes the violation ϕ of constraint (14.4) and identifies the unsafe components which still have a residual capacity (the ones with $\tau_\ell < w(F'_\ell)$), along with the minimal value μ of these capacities. As long as the violation cannot be fairly divided among the components without exceeding their residual capacity, we increase σ and τ_ℓ (for all ℓ such that $\tau_\ell < w(F'_\ell)$) by μ to keep

```
Input  : $G = (V, E)$: connected undirected graph
         $w : V \to \mathbb{R}^+$: weight function on the vertices
         $\langle S, U, F\rangle$ current sub-problem
1  Algorithm Strict Relaxation($G, w, S, U, F$):
2      $z := \max\left\{ w(S_k),\ \max_{\ell \in L'} w(U_\ell) \right\}$
3      $\sigma := z - w(S_k)$                              // Satisfies constraints(14.2)
4      $\tau_\ell := \min\{w(F'_\ell),\ z - w(U_\ell)\}$   $\forall \ell \in L'$  // Keeps satisfied constraints(14.3)
5      loop
6          $\phi := w(F') - \sigma - \sum_{\ell \in L'} \tau_\ell$     // $\phi$ is the violation of constraint (14.4)
7          $Q := \{\ell \in L' \mid \tau_\ell < w(F'_\ell)\}$          // Set of all components
                                                                     with residual capacity
8          $\bar\phi := \phi/(|Q|+1)$
9          $\mu := \min_{\ell \in Q} \{w(F'_\ell) - \tau_\ell\}$         // Minimal residual capacity among $Q$
                                                                     ($\mu = +\infty$ if $Q = \emptyset$)
10         if $\bar\phi \le \mu$ then exit loop
11         for $\ell \in Q$ do $\tau_\ell := \tau_\ell + \mu$            // Since $\forall \ell \in Q : \mu \le w(F'_\ell) - \tau_\ell$,
                                                                     then (14.3) are satisfied
12         $\sigma := \sigma + \mu$ // Since the increment is the same, (14.2) are satisfied
13     end
14     if $\phi > 0$ then    // When (14.4) was not already satisfied before the loop
15         $\sigma := \sigma + \bar\phi$
16         $\tau_\ell := \tau_\ell + \bar\phi$   $\forall \ell \in Q$        // $\bar\phi \le \mu$ so the increment respects (14.3)
17     end
18     return $\sigma$,   $\forall \ell \in L'$ : $\tau_\ell$
```

Algorithm 1: Algorithm to solve problem (14)

constraints (14.3) satisfied. Increasing them by the same amount, also constraints (14.2) remain satisfied. When the condition on the residual violation no longer holds, we exit the loop and divide the violation equally among the remaining components, to minimise the increase of the objective function. If constraint (14.4) was already satisfied before the loop, it means that we already have the optimal solution and that we must not decrease the variables by $\bar\phi \le 0$ because it could possibly violate constraints (14.5).

At each iteration, at least one of constraints (14.3) is activated, implying that, after at most $|L'|$ iterations, $Q = \emptyset$ and therefore $\mu = +\infty \ge \bar\phi$, exiting the loop. Once outside the loop, $\bar\phi \le \mu$, so $\tau_\ell + \bar\phi \le w(F'_\ell)$ for all $\ell \in Q$: it is possible to increase σ and every τ_ℓ with $\ell \in Q$ by $\bar\phi$. Increasing $|Q|+1$ variables by $\bar\phi = \phi/(|Q|+1)$ reduces the violation of (14.4) to zero, thus producing a feasible solution.

To prove that Algorithm 1 returns the optimal solution, let us recall that throughout its execution, constraints (14.2) and (14.3) are never violated and that the termination condition precisely corresponds to the satisfaction of constraint (14.4). Therefore, Algorithm 1 returns a feasible solution of formulation (14). To prove that the said solution is also optimal we first consider the following lemma.

Lemma 2. *Let σ^*, τ^*_ℓ be the solution returned by Algorithm 1, for each $\ell \in L'$ such that $\tau^*_\ell < w(F'_\ell)$, constraints* (14.2) *are satisfied with equality.*

$$\forall \ell \in L' : \tau^*_\ell < w(F'_\ell) \implies w(S_k) + \sigma^* = w(U_\ell) + \tau^*_\ell$$

Proof. The condition holds at every iteration of the algorithm and we prove it by induction on the number of iterations performed. The base case is right before the beginning of the loop. If $\tau_\ell < w(F'_\ell)$ then $\tau_\ell = z - w(U_\ell)$ which implies that

$$w(U_\ell) + \tau_\ell = w(U_\ell) + z - w(U_\ell) = z = w(S_k) + \sigma$$

Now, the inductive hypothesis is that at the beginning of the iteration, if $\tau_\ell < w(F'_\ell)$ then $w(S_k) + \sigma = w(U_\ell) + \tau_\ell$. Since $\tau_\ell < w(F'_\ell)$ then $\ell \in Q$ and therefore, at the end of the iteration, τ_ℓ is incremented by μ. We know that also σ is increased by μ therefore

$$w(S_k) + \sigma = w(U_\ell) + \tau_\ell \iff w(S_k) + \sigma + \mu = w(U_\ell) + \tau_\ell + \mu$$

so that at the beginning of the next iteration the condition will still hold. Once we exit the loop, all variables with residual capacities are incremented by the same amount, thus maintaining the condition satisfied. □

Now we prove the optimality.

Theorem 2. *The solution σ^*, τ^*_ℓ returned by Algorithm 1 is optimal.*

Proof. Without loss of generality we assume that $\sigma^* \leq w(F')$, satisfying the condition of Remark 2. This assumption is justified, as if $\sigma^* > w(F')$ it would mean that $\phi < 0$ which is only possible when at the beginning of the algorithm σ^* is set to $z - w(S_k)$. This happens when there is some unsafe component U_ℓ of weight $z > w(S_k)$ and, therefore, the minimum σ such that $w(S_k) + \sigma \geq w(U_\ell)$ is $\sigma^* = w(U_l) - w(S_k) = z - w(S_k)$, proving that σ^*, τ^*_ℓ is optimal. On the other hand, if $\sigma^* \leq w(F')$ we can use the procedure in Remark 2 to find an equivalent solution that satisfies constraint (14.4) to equality.

By contradiction, there is some feasible solution $\bar\sigma, \bar\tau_\ell$ of formulation (14) such that $\bar\sigma < \sigma^*$. Once again, without loss of generality, we assume $\bar\sigma, \bar\tau_\ell$ to satisfy constraint (14.4) to equality (notice that $\bar\sigma < \sigma^* \leq w(F')$). Since both are feasible solutions, satisfying constraint (14.4) to equality, the following equation holds

$$\sigma^* + \sum_{\ell \in L'} \tau^*_\ell = w(F') = \bar\sigma + \sum_{\ell \in L'} \bar\tau_\ell$$

but since $\bar\sigma < \sigma^*$

$$\sum_{\ell \in L'} \tau^*_\ell < \sum_{\ell \in L'} \bar\tau_\ell$$

which means that there exists some $\ell^+ \in L'$ such that $\tau^*_{\ell^+} < \bar\tau_{\ell^+} \leq w(F'_{\ell^+})$.
By Lemma 2 we know that $w(S_k) + \sigma^* = w(U_{\ell^+}) + \tau^*_{\ell^+}$ therefore

$$w(S_k) + \bar\sigma < w(S_k) + \sigma^* = w(U_{\ell^+}) + \tau^*_{\ell^+} < w(U_{\ell^+}) + \bar\tau_{\ell^+}$$

violating constraint (14.2). □

In Figure 3.4, $F' = \{1,10,11\}$ is partitioned assigning vertex 1 to U_1, vertex 10 to U_4 and vertex 11 to U_3, while F'_2 remains empty because U_2 has a weight smaller than U_3, even though it is adjacent to vertex 11. Now, $z = w(U_1) = 15$, so we need to increase the weights of the other components to that value. This amounts to setting $\sigma = 7$, $\tau_3 = 5$ and $\tau_4 = 2$, while $\tau_1 = \tau_2 = 0$ because U_1 already has the maximum weight and U_2 cannot increase its weight ($F'_2 = \emptyset$). The residual violation to be distributed is $\phi = w(F') - \sigma - \sum_\ell \tau_\ell = 29 - 7 - 5 - 2 = 15$ and $Q = \{1,3,4\}$ identifies the unsafe components with residual capacity; the minimum capacity is $\mu = w(F'_1) - \tau_1 = 2$. Since $\bar{\phi} > \mu$, we increase all variables τ_ℓ with $\ell \in Q$ and σ by μ, setting $\tau_1 = 2$, $\tau_3 = 7$, $\tau_4 = 4$, and $\sigma = 9$. This decreases the residual violation to $\phi = 7$ and leaves only 2 augmentable components ($Q = \{3,4\}$). As $\mu = w(F'_3) - \tau_3 = w(F'_4) - \tau_4 = 8$, and it is not smaller than $\bar{\phi} = 7/3$, we exit the loop and distribute the residual violation in equal parts among σ, τ_3 and τ_4, increasing them by $7/3$. Consequently, the optimal value of σ is $\sigma^* = 9 + 7/3$ and the lower bound is $w(S) + \sigma^* = 15 + 8 + 9 + 7/3 = 34 + 1/3$, that can be rounded up to 35 (since all weights are integers). This is much larger than the two simple bounds, that are, respectively, equal to $w(S) = 23$ and $w(U_1) = 15$. The weight of the optimal solution $S^* = \{2,4,6,8,11\}$ is 42.

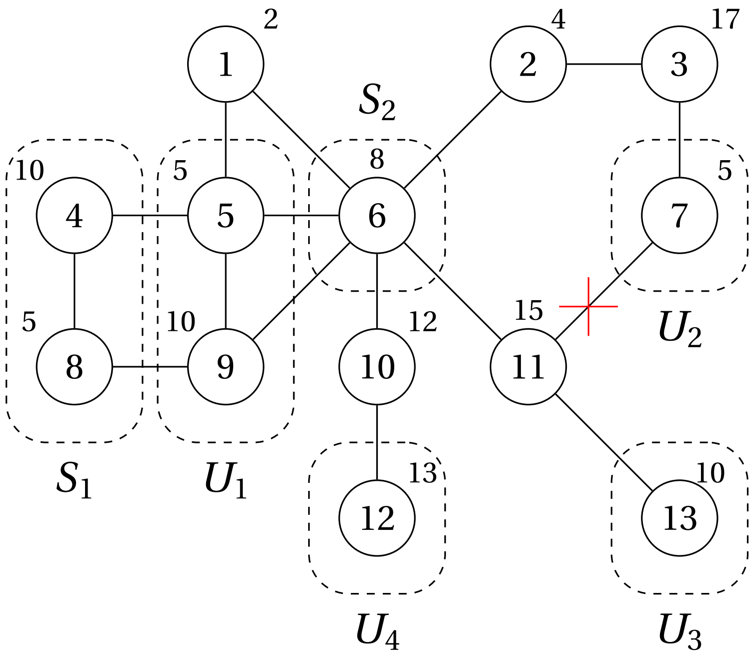

Figure 3.4: Example of the computation of the strict lower bound for a sub-problem $\langle S, U, F\rangle$: the current total weight of the safe components is $w(S) = 23$ and the current maximum weight of the unsafe components is $w(U_1) = 15$, but the lower bound can be raised up to 35, partitioning $F' = \{1,10,11\}$ into $F'_1 = \{1\}$, $F'_2 = \emptyset$, $F'_3 = \{11\}$ and $F'_4 = \{10\}$ and distributing its weight among S_2, U_3 and U_4. This strongly reduces the gap with respect to the optimal solution $S^* = \{2,4,6,8,11\}$, whose weight is 42.

The complexity of Algorithm 1 is easily identified by acknowledging that the main loop decreases the cardinality of Q by at least 1 in each iteration, resulting in a maximum of $|\mathscr{C}_G(U)| \le |V|$ iterations. Since each iteration requires to scan the unsafe components to update Q, compute μ and possibly update variables τ_ℓ, it has complexity $O(|\mathscr{C}_G(U)|) \le |V|$. Overall, the complexity of the algorithm is $O(|\mathscr{C}_G(U)|^2) \le O(|V|^2)$.

3.2.2 Flexible lower bound

We introduce another property that allows to manipulate the graph obtaining a relaxation of sub-problem $\langle S, U, F\rangle$.

Proposition 5. *Adding to graph $G = (V, E)$ an edge (u, v) with $u, v \in S$ provides a relaxation of sub-problem $\langle S, U, F\rangle$.*

Proof. It is a relaxation because all feasible solutions of the sub-problem keep their value and remain feasible in the modified graph. Given two fixed safe vertices $u, v \in S$ and a solution $\tilde{S}$ for sub-problem (S, U, F), there are two cases:

1. u and v belong to the same safe component induced by $\tilde{S}$: then, adding edge (u, v) changes neither the components nor the value of $\tilde{S}$;

2. u and v belong to different safe components induced by $\tilde{S}$: the addition of (u, v) merges the two safe components, leaving unchanged all unsafe components and the value of the objective.

$\square$

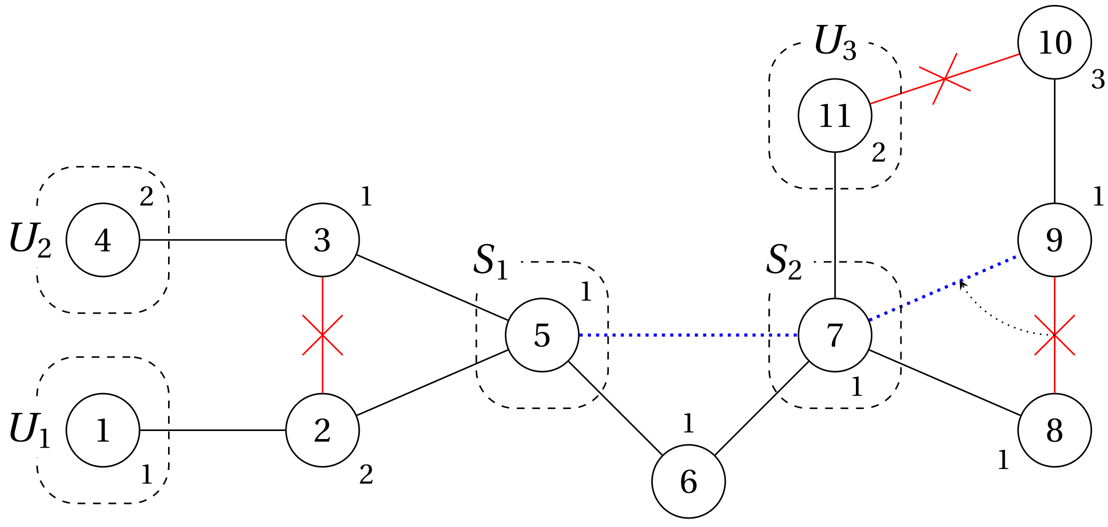

Figure 3.5: Example of graph manipulations: inserting edge $(5,7)$ (in dotted blue), removing the edges $(2,3), (10,11)$ (both in red) or replacing edge $(8,9)$ (also in red) with $(7,9)$ (in dotted blue) yields a relaxation of the original problem.

Figure 3.5 shows an example of usage of the three properties (3, 4, 5) similar to Figure 3.3. We consider the sub-problem in which the vertex set $V = \{1, \ldots, 11\}$ is partitioned into $S = \{5,7\}$, $U = \{1,4,11\}$ and $F = \{2,3,6,8,9,10\}$. Dashed boxes mark the two safe components $S_1 = \{5\}$ and $S_2 = \{7\}$, and the three unsafe components $U_1 = \{1\}$, $U_2 = \{4\}$ and $U_3 = \{11\}$, while the frontier is $F' = \{2,3\}$. Table 3.2 provides examples for the application of the previous propositions, reporting the kind of modification, the corresponding proposition, the optimal solution of the relaxed instance obtained (S^*) and the value of the optimum ($w(S^*)$). The first row refers to the starting graph with no modification, the last one to the graph obtained applying all of them. Figure 3.5 marks with a cross a removed or replaced edge and draws with a dotted line a new edge; in case of replacement, a line with an arrowhead connects the original edge with the new one. In all examples, the optimal solution of the original problem remains feasible, but a new better solution appears.

Modification	Proposition	S^*	$w(S^*)$
none		$\{3,5,6,7,8,9\}$	6
insert $(5,7)$	5	$\{3,5,7,8,9\}$	5
remove $(2,3)$	3	$\{5,6,7,8,9\}$	5
remove $(10,11)$	3	$\{3,5,6,7,8\}$	5
replace $(8,9)$ with $(7,9)$	4	$\{3,5,6,7,9\}$	5
all the previous	5, 3, 4	$\{5,7,9\}$	3

Table 3.2: Manipulations applied to the graph in Figure 3.5 and their effects on the optimal solution.

The lower bounding procedure applies all the manipulations of the strict version but also adds an initial step where all the safe vertices are connected to each other. Once again, the following manipulations allow to modify graph G to obtain an auxiliary graph G' as follows:

- connect all vertices in S to one another;
- remove all edges (u, v) with $u, v \in F'$;
- remove all edges (u, v) with $u \in U$ and $v \in F \setminus F'$;
- for each vertex $v \in F'$, remove all but one of the edges (u, v) such that $u \in U$;
- replace every edge (u, v) with $u \in F'$ and $v \in F \setminus F'$ with an edge (s, v) with $s \in S$ (if such an edge already exists, just remove (u, v)).

The optimal solution of sub-problem $\langle S, U, F\rangle$ on the auxiliary graph G' is smaller than or equal to the optimal solution for G. In order to find it, we can exploit the main features of G':

- the safe vertices form a single safe component;
- the unsafe components remain unchanged: $\mathscr{C}_G(U) = \mathscr{C}_{G'}(U)$
- the unsafe components can include vertices of F', but not of $F \setminus F'$, and they cannot merge;
- each $v \in F'$ joins either S or the only unsafe component adjacent to it.

Just like for the Strict lower bound, the choice of the edge between each $v \in F'$ and U is heuristic and we associate v with the adjacent unsafe component of maximum weight.

This time, we partition the frontier F' in a slightly different way. In fact, instead of considering only the unsafe components adjacent to S or to F', we take into consideration all unsafe components. This is because including all unsafe components introduces more constraints, leading to a stronger lower bound. Additionally, since we only consider a single safe component, all unsafe components will be adjacent to the same safe component in any solution. Therefore we redefine F'_ℓ as follows:

$$F'_\ell = \{v \in F' \mid U_\ell = \arg \max_{\substack{U_y \in \mathscr{C}_G(U): \\ U_y \bowtie_G \{v\}}} w(U_y)\} \qquad U_\ell \in \mathscr{C}_G(U)$$

where F'_ℓ is empty when the unsafe component U_ℓ is not adjacent to F' in G'.

The new *ILP* formulation for sub-problem $\langle S, U, F\rangle$ on graph G' is

$$\min w(S) + \sigma \tag{15.1}$$

subject to:

$$w(S) + \sigma \geq w(U_\ell) + \tau_\ell \qquad U_\ell \in \mathscr{C}_G(U) \tag{15.2}$$

$$\sigma = \sum_{v \in F} w_v \cdot x_v \tag{15.3}$$

$$\tau_\ell = \sum_{v \in F'_\ell} w_v \cdot (1 - x_v) \qquad U_\ell \in \mathscr{C}_G(U) \tag{15.4}$$

$$\sigma \geq 0 \tag{15.5}$$

$$\tau_\ell \geq 0 \qquad U_\ell \in \mathscr{C}_G(U) \tag{15.6}$$

$$x_v \in \{0, 1\} \qquad v \in F \tag{15.7}$$

Formulation (15) shares its structure with formulation (11) with the main difference that, this time, the safe component considered is the whole set of vertices fixed to safe (because we connected them) and all unsafe components are considered.

As before, since the formulation (15) is difficult to solve, we replace the constraints (15.4) and (15.3) with properties (12) and (13), resulting in the new continuous LP formulation (16).

$$\min\ w(S) + \sigma \tag{16.1}$$

subject to:

$$w(S) + \sigma \geq w(U_\ell) + \tau_\ell \qquad U_\ell \in \mathscr{C}_G(U) \tag{16.2}$$

$$\sigma + \sum_{U_\ell \in \mathscr{C}_G(U)} \tau_\ell \geq w(F') \tag{16.3}$$

$$\sigma \geq 0 \tag{16.4}$$

$$0 \leq \tau_\ell \leq w(F'_\ell) \qquad U_\ell \in \mathscr{C}_G(U) \tag{16.5}$$

And solve it by slightly changing Algorithm 1 into Algorithm 2.

```
Input : $G = (V, E)$: connected undirected graph
        $w : V \to \mathbb{R}^+$: weight function on the vertices
        $\langle S, U, F \rangle$ current sub-problem
Algorithm Flexible Relaxation($G, w, S, U, F$):
    $z = \max\{w(S), \max_{U_\ell \in \mathscr{C}_G(U)} w(U_\ell)\}$
    $\sigma = z - w(S)$
    for $U_\ell \in \mathscr{C}_G(U)$ do $\tau_\ell = \min\{w(F'_\ell), z - w(U_\ell)\}$
    loop
        $\phi = \max\{w(F') - \sigma - \sum_{U_\ell \in \mathscr{C}_G(U)} \tau_\ell, 0\}$
        $Q = \{U_\ell \in \mathscr{C}_G(U) \mid \tau_\ell < w(F'_\ell)\}$
        $\bar{\phi} = \phi / (|Q| + 1)$
        $\mu = \min_{U_\ell \in Q} \{w(F'_\ell) - \tau_\ell\}$
        if $\bar{\phi} \leq \mu$ then exit loop
        $\sigma = \sigma + \mu$
        for $U_\ell \in \mathscr{C}_G(U)$ do $\tau_\ell = \tau_\ell + \mu$
    end
    $\sigma = \sigma + \bar{\phi}$
    for $U_\ell \in \mathscr{C}_G(U)$ do $\tau_\ell = \tau_\ell + \bar{\phi}$
    return $w(S) + \sigma$
```

Algorithm 2: Exact algorithm to solve formulation (16) for sub-problem $\langle S, U, F \rangle$.

Consider the example of Figure 3.5: after applying all graph manipulations, $w(S) = 2$, $w(U_1) = 1$, $w(U_2) = 2$, $w(U_3) = 2$, $w(F'_1) = 2$, $w(F'_2) = 1$, $w(F'_3) = 0$. The initialisation sets $z = w(S) = 2$, $\tau_1 = 0$, $\tau_2 = 1$ and $\tau_3 = 0$. In the first iteration of the loop, $\phi = 2$ and $Q = \{U_1, U_2\}$, so $\bar{\phi} = 2/3$ and $\mu = 1$. We can apply the increase to all components and raise z to $2 + 2/3$, consistently with the fact that the optimum for graph G' is 3.

The correctness of Algorithm 2 can be proven in a similar manner as Algorithm 1 and their complexity is exactly the same.

3.3 Vertex fixing

Section 3.1 described a feasibility test for a given sub-problem $\langle S,U,F\rangle$, that also moves some vertices from F to U, based on the fact that any solution including them is necessarily infeasible (see Proposition 2). In the following, we introduce other properties that allow to move free vertices to S or U, thus reducing the size of the current sub-problem. The first one simply extends the feasibility test with an implicit branching operation.

Proposition 6. *Given a sub-problem $\langle S,U,F\rangle$, let C be a component of $\mathscr{C}_G(S \cup F)$ and f a vertex of $C \cap F$. If*

$$w(C) - w_f < \sum_{\substack{U_j \in \mathscr{C}_G(U): \\ U_j \bowtie \{f\}}} w(U_j) + w_f \tag{17}$$

then vertex f belongs to the safe set of all feasible solutions of $\langle S,U,F\rangle$.

Proof. Moving f from F to U modifies sub-problem $\langle S,U,F\rangle$ into sub-problem $(S, U \cup \{f\}, F \setminus \{f\})$. Correspondingly, all the unsafe components $U_j \in \mathscr{C}_G(U)$ that were adjacent to f ($U_j \bowtie \{f\}$) merge into a single one, whose weight is the right-hand-side of (17). On the other hand, component C loses vertex f and possibly splits into several components, each with a weight dominated by the left-hand-side of (17). Therefore, at least one of the safety constraints concerning subset $S \cup F \setminus \{f\}$ is violated. □

The following property assigns a vertex with another implicit branching based on the comparison with the value of the best known solution, $\bar{S}$, that is an upper bound on the optimum.

Proposition 7. *Let $\langle S,U,F\rangle$ be a sub-problem and $f \in F$ a free vertex. If*

$$w(S) + w_f \geq w(\bar{S})$$

then vertex f does not belongs to the safe set of any feasible solution better than $\bar{S}$. If

$$\sum_{\substack{U_j \in \mathscr{C}_G(U): \\ U_j \bowtie \{f\}}} w(U_j) + w_f \geq w(\bar{S})$$

then vertex f belongs to the safe set of all feasible solutions better than $\bar{S}$.

Proof. The proof is based on computing the two simple lower bounds described in the beginning of Section 3.2, respectively assuming that vertex f is moved to S or to U. □

Notice that moving free vertices to S can create new safe components, possibly violating the assumption made in Section 3.2.1 for the Strict lower bound, that at most one safe component is adjacent to F. Since this assumption is fundamental for the correct execution of the lower bound procedure, we move into S only free vertices that are already adjacent to S. On the other hand, if we opt for the Flexible lower bound in Section 3.2.2, we can apply all the fixings without any limitation.

3.4 Exploration strategy and branching rule

We adopted a *best-first* strategy for the visit of the branching tree, visiting first the open node which minimises the lower bound, in order to focus the search on the most promising nodes and produce good upper bounds.

For what concerns the branching strategy, we select the branching vertex as the free vertex of maximum weight. The rationale of this choice is that assigning such a vertex in both ways could increase the lower bound by increasing either the total weight of the safe vertices or the weight of the heaviest unsafe component. In case of ties, we consider the vertex of maximum degree because it is more likely to be involved in a larger number of safety constraints, thus strengthening the lower bound.

As already mentioned above, moving free vertices to S can violate the basic condition required to compute the Strict lower bound. To avoid doing that, we modify the branching rule selecting the free vertex of maximum weight and degree only among those that are adjacent to S. In case no such vertex exists, we select it among those that are adjacent to U. At the root of the branching tree, where both S and U are empty, we simply apply the basic branching rule. It is worth mentioning that the modified branching rule also allows to apply the algorithm to the *WCSSP*. In order to maintain a single safe component, in fact, it is possible to restrict the selection of the branching vertex only to the free vertices adjacent to it. In case no free vertex is adjacent to the safe component, all remaining free vertices must be moved to the unsafe set. This either solves the current node or proves that it is unfeasible.

For the Flexible lower bound, however, we just consider the free vertex of maximum weight and, in case of ties, of maximum degree.

3.5 Experimental results

In this section, we analyse the performances of the two versions of the branch-and-bound algorithm: with strict relaxation (and weaker vertex fixings and branching rule) or with flexible relaxation (and more effective vertex fixings and branching rule). At first we compare the state of the art approaches to the version with strict relaxation and, then, we show that the version with flexible lower bound outperforms the one with strict lower bound, and therefore all the other methods.

The instances provided by the literature are the ones of benchmarks `M` and `H` with number of vertices $|V| \leq 50$. Here we simply extend benchmark `H` introducing instances with $|V| = 60$. The other benchmarks include instances too large to produce meaningful results.

All these approaches run on different, but comparable, machines. Table 3.3 reports the acronyms used in the following to denote them and their main features. The number of threads (Thr.) for B&C1 is not explicitly reported, but is presumably equal to 1. The following additional information is available on the solvers and options. B&C1 was implemented in C++ using CPLEX 12.6 and disabling all heuristics, cuts and pre-processing. The MILP formulation by Hosteins (2020) was solved with CPLEX 12.9, disabling CPLEX's cuts, that slowed down excessively the solution process. B&C2 was written in C/C++ and compiled with g++ version 8.3.0, exploiting the Solving Constrained Integer Programming Optimization Suite version 6.0.2 along with the Soplex LP solver version 4.0.2; a test with Gurobi reduced the number of branching nodes, but increased the overall computational time.

Name	Reference	Processor	Frequency	RAM	Thr.
B&C1	Macambira et al. (2019)	Intel Core TM i7-6700	3.40 GHz	15.6 GB	-
MILP	Hosteins (2020)	Intel Core i7-6600U	2.60 GHz	32 GB	4
B&C2	Malaguti and Pedrotti (2023)	Intel Core i7-4790	3.60 GHz	32 GB	1
B&Bs	This contribution (Strict)	Intel Xeon E5-2620	2.1 GHz	16 GB	1
B&Bv	This contribution (Flexible)	Intel Xeon E5-2620	2.1 GHz	16 GB	1

Table 3.3: Details of the exact algorithms for the *WSSP* and machines used to test them.

Definition of the total optimality gap The quality of the solution returned by an exact algorithm is usually measured with the *total optimality gap* defined as

$$(UB - LB)/z^*$$

where

- z^* is the value of the optimal solution to the problem;
- UB is the upper bound returned by the algorithm;
- LB is the lower bound returned by the algorithm.

Usually, z^* is unknown and, therefore we have to consider an estimate. The literature often consider two:

$$(UB - LB)/UB \tag{18}$$
$$(UB - LB)/LB \tag{19}$$

and the following relationship holds

$$(UB - LB)/UB \leq (UB - LB)/z^* \leq (UB - LB)/LB$$

In the following, when possible (in particular, when comparing the different versions of our algorithm) we will prefer the overestimate (19). However, when comparing our results to those of authors who adopt the underestimate, for the sake of consistency we will adopt (18).

3.5.1 Comparison with the state of the art

We compared the results of B&Bs with those of the competing algorithms available in the literature.

Table 3.4 reports the results for the weighted benchmark `M`. Each row refers to a value of size n, that is reported in the first column. The following three groups of four columns report the computational time in seconds required by the four approaches to solve the three instances of size n, which differ by the density δ of the graph. These results derive from the original references, with the original precision. The instances with a computational time of 7200 seconds are actually not solved to optimality because of the imposition of a time limit of 2 hours. While B&C2 dominates the other approaches for $\delta = 0.3$, B&Bs is slightly slower on those instances, slightly faster for $\delta = 0.5$ and two to three orders of magnitude faster for $\delta = 0.7$. The computational time of MILP (some minutes) and B&C1 (sometimes more than 2 hours) are three to four orders of magnitude larger.

	$\delta = 0.3$				$\delta = 0.5$				$\delta = 0.7$			
$\|V\|$	B&C1	MILP	B&C2	B&Bs	B&C1	MILP	B&C2	B&Bs	B&C1	MILP	B&C2	B&Bs
10	0.87	0	0.0	0.000	1.58	0	0.0	0.001	1.28	1	0.0	0.000
11	0.70	1	0.0	0.000	1.19	0	0.1	0.000	2.78	1	1.4	0.000
12	2.71	1	0.0	0.001	4.13	0	0.1	0.000	4.05	1	7.3	0.001
13	1.93	1	0.0	0.000	3.92	1	0.0	0.000	4.55	1	1.6	0.001
14	6.55	2	0.0	0.001	8.15	2	0.1	0.001	13.78	3	2.7	0.001
15	8.14	2	0.0	0.001	20.73	3	0.1	0.002	26.66	4	2.6	0.003
16	30.72	4	0.0	0.001	27.11	4	0.2	0.001	45.04	6	7.2	0.002
17	24.85	4	0.0	0.003	45.35	4	0.1	0.004	58.95	4	4.3	0.003
18	35.05	4	0.0	0.001	43.11	7	0.5	0.002	96.20	10	2.8	0.003
19	72.18	13	0.0	0.007	74.87	10	0.4	0.003	158.06	14	5.8	0.008
20	130.58	11	0.0	0.009	126.04	13	0.2	0.005	317.75	19	25.4	0.011
21	222.34	11	0.0	0.006	326.43	20	0.5	0.031	304.43	33	9.6	0.036
22	863.93	22	0.0	0.028	515.10	17	0.1	0.032	1422.93	51	1.9	0.028
23	243.63	19	0.1	0.009	428.23	26	0.7	0.012	661.38	44	12.0	0.008
24	455.48	69	0.0	0.040	810.46	59	1.6	0.022	2296.73	76	7.2	0.050
25	940.33	61	0.1	0.046	2209.33	49	1.2	0.046	3065.93	50	13.3	0.029
26	2892.70	203	0.1	0.131	3042.08	80	1.4	0.109	3167.12	113	58.4	0.041
27	2648.34	71	0.1	0.127	4399.39	83	1.6	0.226	2626.44	262	26.6	0.075
28	3081.24	221	0.0	0.100	5961.68	282	0.4	0.084	6761.96	251	14.7	0.051
29	5559.33	138	0.0	0.416	5574.58	211	0.8	0.469	7200.00	254	65.9	0.241
30	3907.08	219	0.2	0.328	7200.00	267	0.1	0.456	7200.00	305	46.8	0.423

Table 3.4: Computational times (in seconds) required to solve the weighted instances of benchmark `M` of density $\delta \in \{0.3, 0.5, 0.7\}$ with B&C1, MILP, B&C2 and B&Bs.

Table 3.5 reports the corresponding results for the unweighted instances. The structure of the table is the same, and the results are consistent with the ones of Table 3.4, except for the fact that the difference between B&C2 and B&Bs are more marked (in favour of B&C2 for $\delta = 0.3$ and of B&Bs for $\delta = 0.5$) and that for $\delta = 0.7$ the computational time of B&C2 becomes higher than that of MILP in most instances.

	$\delta = 0.3$				$\delta = 0.5$				$\delta = 0.7$			
$\lvert V\rvert$	B&C1	MILP	B&C2	B&Bs	B&C1	MILP	B&C2	B&Bs	B&C1	MILP	B&C2	B&Bs
10	0.58	1	0.0	0.000	0.68	0	0.0	0.000	0.72	1	0.0	0.000
11	1.02	0	0.0	0.000	0.94	1	0.2	0.000	1.40	0	19.6	0.000
12	1.58	1	0.0	0.000	2.90	1	0.1	0.001	3.52	1	4.3	0.000
13	1.59	1	0.0	0.001	3.59	1	0.0	0.001	4.34	1	27.5	0.001
14	7.72	2	0.0	0.001	4.60	2	0.2	0.002	6.67	2	38.8	0.001
15	10.08	2	0.0	0.004	14.80	3	0.3	0.002	15.17	4	10.1	0.003
16	16.65	2	0.0	0.004	25.44	4	0.8	0.007	32.52	6	33.1	0.002
17	24.36	5	0.0	0.006	47.98	6	1.5	0.004	48.90	4	455.9	0.002
18	32.09	7	0.0	0.005	39.22	9	2.6	0.003	80.11	6	80.7	0.002
19	54.67	10	0.0	0.017	84.42	10	0.5	0.019	101.36	12	24.9	0.015
20	231.73	11	0.0	0.037	222.89	15	0.8	0.019	194.30	22	93.3	0.004
21	297.90	16	0.0	0.014	408.62	27	0.7	0.016	190.03	37	140.0	0.002
22	379.42	18	0.0	0.033	714.91	15	1.0	0.040	750.14	27	52.1	0.007
23	614.10	34	0.0	0.050	467.30	41	3.7	0.057	546.10	94	418.1	0.018
24	381.10	38	0.0	0.107	658.42	58	0.2	0.155	2434.47	110	311.0	0.012
25	1090.78	59	0.1	0.216	907.91	44	0.5	0.097	1768.22	123	80.7	0.079
26	1560.85	78	0.1	0.267	3307.20	97	6.5	0.182	2388.87	148	878.8	0.017
27	1853.99	150	0.0	0.240	2462.43	134	3.6	0.090	3899.18	168	2416.3	0.186
28	4557.31	284	0.1	0.421	3891.22	248	3.6	0.218	2708.83	194	111.2	0.036
29	4276.96	182	0.1	0.494	4089.23	130	11.1	0.303	4679.83	178	2418.2	0.516
30	4962.59	472	0.1	1.723	7200.00	359	11.1	0.549	7200.00	332	439.1	0.053

Table 3.5: Computational times (in seconds) required to solve the unweighted instances of benchmark M of density $\delta \in \{0.3, 0.5, 0.7\}$ with B&C1, MILP, B&C2 and B&Bs.

Table 3.6 compares the results of MILP, B&C2 and B&Bs on benchmark H for the weighted instances. These are available in the literature only for $|V| \leq 50$. Each row of the table refers to a given density and size, reported in the first two columns. The table reports the arithmetic mean of the optimality gap on each group of 5 instances thus identified, the number of instances solved within the time limit of one hour and the average computational time in seconds. For consistency with the results published in Malaguti and Pedrotti (2023), we compute the optimality gap obtained by each algorithm a on an instance as the difference between the upper and the lower bound divided by the upper bound, that is $(UB_a - LB_a)/UB_a$. This provides a lower estimate of the actual optimality gap, that would be $(UB_a - LB_a)/z^*$, where z^* denotes the optimal value.

The proposed branch-and-bound algorithm solves all 120 instances exactly (in 40 minutes in the worst case), whereas MILP solves 71 instances up to 40 vertices and B&C2 solves 97 instances up to 40 vertices. On the biggest instances, the optimality gaps are around 20 – 30% for B&C2, larger than 90% for MILP.

		MILP			B&C2			B&Bs		
δ	$\|V\|$	gap	solved	sec	gap	solved	sec	gap	solved	sec
0.1	20	-	5	72.2	-	5	0.8	-	5	0.002
	25	-	5	746.4	-	5	4.9	-	5	0.009
	30	16.65	3	2554.6	-	5	20.1	-	5	0.046
	35	39.53	0	3600.0	-	5	55.0	-	5	0.147
	40	74.89	0	3600.0	2.2	4	1244.2	-	5	4.306
	50	94.55	0	3600.0	22.7	0	3600.0	-	5	237.367
0.2	20	-	5	28.0	-	5	0.3	-	5	0.008
	25	-	5	140.2	-	5	1.9	-	5	0.138
	30	7.52	4	1472.0	-	5	10.6	-	5	0.803
	35	34.49	1	3522.6	-	5	195.3	-	5	6.127
	40	72.21	0	3600.0	-	5	1207.8	-	5	21.879
	50	93.72	0	3600.0	33.1	0	3600.0	-	5	1072.342
0.3	20	-	5	11.8	-	5	0.2	-	5	0.013
	25	-	5	52.0	-	5	2.3	-	5	0.114
	30	-	5	562.2	-	5	22.5	-	5	0.481
	35	4.31	4	1956.8	-	5	326.6	-	5	3.133
	40	43.19	1	3384.0	2.1	3	2400.3	-	5	10.821
	50	96.36	0	3600.0	24.1	0	3600.0	-	5	90.216
0.4	20	-	5	12.8	-	5	0.5	-	5	0.016
	25	-	5	66.4	-	5	5.1	-	5	0.088
	30	-	5	375.0	-	5	34.2	-	5	0.465
	35	-	5	905.4	-	5	404.6	-	5	2.151
	40	16.05	3	2788.0	-	5	1367.2	-	5	4.239
	50	99.41	0	3600.2	17.8	0	3600.0	-	5	50.748

Table 3.6: Average gap (%), number of instance solved within an hour and average computational time (in seconds) required to solve the weighted instances of benchmark H with $|V| \leq 50$ of density $\delta \in \{0.1, 0.2, 0.3, 0.4\}$ with MILP, B&C2 and B&Bs.

The results for the unweighted instances of the same benchmark are displayed in Table 3.7 and present similar features. B&Bs closes 115 instances over 120, exhausting the available memory in less than one hour on the 5 instances with $n = 50$ and $\delta = 0.1$. In this case, B&C2 and MILP have a similar performance: 73 solved instances for MILP versus 74 for B&C2, with a predominance of the former on the dense instances and of the latter on the sparse ones, and shorter computational times for B&C2 on the smaller instances.

		MILP			B&C2			B&Bs		
δ	$\|V\|$	gap	solved	sec	gap	solved	sec	gap	solved	sec
0.1	20	-	5	94.2	-	5	1.3	-	5	0.020
	25	-	5	532.2	-	5	5.2	-	5	0.042
	30	4.44	4	1743.8	-	5	32.4	-	5	0.189
	35	33.84	1	3538.0	-	5	469.4	-	5	3.943
	40	67.73	0	3600.0	19.0	1	2976.1	-	5	107.544
	50	87.97	0	3600.0	37.6	0	3600.0	35.78	0	1716.793
0.2	20	-	5	15.4	-	5	0.6	-	5	0.034
	25	-	5	234.4	-	5	4.9	-	5	0.434
	30	-	5	856.8	-	5	45.2	-	5	1.801
	35	18.01	2	3437.0	7.7	2	2825.1	-	5	20.079
	40	71.13	0	3600.0	29.2	0	3116.9	-	5	55.688
	50	93.11	0	3600.0	45.8	0	3600.0	-	5	2500.817
0.3	20	-	5	14.4	-	5	0.2	-	5	0.016
	25	-	5	77.6	-	5	12.5	-	5	0.193
	30	-	5	362.6	-	5	174.3	-	5	1.069
	35	10.64	3	2798.2	10.5	1	3141.5	-	5	6.962
	40	38.39	0	3600.0	30.3	0	3368.4	-	5	29.723
	50	91.05	0	3600.0	43.2	0	3600.0	-	5	873.641
0.4	20	-	5	14.2	-	5	1.1	-	5	0.020
	25	-	5	98.8	-	5	40.8	-	5	0.166
	30	-	5	267.8	-	5	256.6	-	5	0.820
	35	2.4	4	1709.4	9.8	1	3486.8	-	5	5.449
	40	11.58	4	2678.4	26.6	0	3600.0	-	5	17.279
	50	95.27	0	3600.0	39.6	0	3600.0	-	5	214.247

Table 3.7: Average gap (%), number of instance solved within an hour and average computational time (in seconds) required to solve the unweighted instances of benchmark H with $|V| \leq 50$ of density $\delta \in \{0.1, 0.2, 0.3, 0.4\}$ with MILP, B&C2 and B&Bs.

Extension of benchmark H In order to check the practical limits of the proposed approach, we extended the original benchmark H of Hosteins (2020), generating in the same way instances with 60 vertices. Table 3.8 displays detailed results for such instances. Each instance is defined by its density δ and an integer index from 1 to 5. Each row displays the results of B&Bs for both the unweighted and weighted version of the problem. We report the upper and lower bound returned by the algorithm on termination (in bold when they coincide), along with the total computation time and the number of branching nodes (BN). When the computer runs out of memory we indicate it with the symbol OM in the BN column. Table 3.8 shows that instances of 60 vertices indeed become challenging. Most weighted instances can still be solved (15 over 20), but four hit the time limit of one hour and one exceeds the available memory in less than one hour. On the other hand, most unweighted instances (16 over 20) cannot be solved: the sparsest ones require too many branching nodes, whereas those with intermediate density reach the time limit with 10% – 25% gaps. Only the densest instances can be solved exactly (or nearly).

Instance		Weighted				Unweighted			
δ	#	UB	LB	sec	BN	UB	LB	sec	BN
0.1	1	**77**	**77**	3474.908	214559949	31	13	1405.774	OM
	2	**79**	**79**	637.409	36033589	34	14	1741.854	OM
	3	**83**	**83**	1130.850	74132535	30	13	842.090	OM
	4	103	84	2641.847	OM	33	13	844.763	OM
	5	**86**	**86**	3486.326	208196017	32	13	1455.969	OM
0.2	1	124	115	3600.000	114519193	30	24	3600.000	124741717
	2	**111**	**111**	818.154	26933287	30	25	3600.000	122956349
	3	145	127	3600.000	118277525	32	24	1864.234	OM
	4	133	130	3600.000	115488031	30	25	3600.000	119412243
	5	138	125	3600.000	110968265	30	25	3600.000	121037137
0.3	1	**129**	**129**	849.539	20954933	30	27	3600.000	89251817
	2	**124**	**124**	500.606	12054917	30	27	3600.000	90466873
	3	**157**	**157**	3002.098	72243237	30	27	3600.000	90122711
	4	**149**	**149**	979.856	22742691	29	28	3600.000	88076127
	5	**144**	**144**	1480.667	36611413	30	27	3600.000	90393441
0.4	1	**137**	**137**	169.997	3122629	**29**	**29**	2178.099	40799191
	2	**130**	**130**	236.432	4292569	**29**	**29**	3097.001	60593849
	3	**163**	**163**	968.583	17609435	**29**	**29**	3199.095	63802285
	4	**154**	**154**	294.560	5287225	**29**	**29**	1597.201	30996299
	5	**155**	**155**	640.122	11745113	30	29	3600.000	71328489

Table 3.8: Upper bound, lower bound, time (in seconds) and number of branching nodes visited by B&Bs to solve unweighted and weighted instances of benchmark H of size $|V| = 60$ and density $\delta \in \{0.1, 0.2, 0.3, 0.4\}$.

3.5.2 Improvements due to the flexible lower bound

We now discuss the impact on the performance of the branch-and-bound algorithm of the flexible lower bounding procedure, along with the resulting more flexible management of the branching tree.

We discuss only very quickly the results on benchmark M. Every instance in this benchmark is solved to optimality in less than half a second. The original algorithm (that was also quite fast) required at most 1.7 seconds. Given these negligible times, a more detailed and careful analysis seems to be of poor significance, so we skip it.

We consider in detail the 280 instances from benchmark H. Most of them can be solved to optimality, but the largest ones are challenging enough to exhaust the available memory or the time limit, that is set to one hour. We compare the results of B&Bv with the ones of B&Bs. Tables 3.9 and 3.10 summarise the comparison, respectively for the weighted and the unweighted instances. The first two columns report the density δ and the size $|V|$ of the instances. Then, two groups of columns provide information on the original branch and bound and the revisited one. Each group includes the average optimality gap, the number of solved instances, the average time consumption in seconds (CPU) and the average number of generated branching nodes (BN). The optimality gap is defined as $(UB - LB)/LB$, where LB is the the lower bound and UB the upper bound produced at the end of the computation, and it is expressed as a percentage. All values are averaged over the 5 instances with the density and size indicated in the associated row. When all 5 instances of a group are solved to optimality, the zero gap is replaced by a dash. An asterisk marks the values for which some instances exhaust the available memory, resulting in an anticipated termination. The symbol OM indicates that all 5 instances of a group went "out of memory".

		B&Bs				B&Bv			
δ	$\|V\|$	gap	solved	CPU	BN	gap	solved	CPU	BN
	20	-	5	0.001	485	-	5	0.003	466
	25	-	5	0.009	2453	-	5	0.007	1830
	30	-	5	0.046	8773	-	5	0.037	6394
0.1	35	-	5	0.147	23993	-	5	0.109	16155
	40	-	5	4.306	518719	-	5	2.788	299276
	50	-	5	237.366	19445646	-	5	111.651	8102922
	60	4.52%	4	2274.268*	133230523*	-	5	1199.706	62877972
	20	-	5	0.009	1570	-	5	0.004	724
	25	-	5	0.138	19199	-	5	0.042	6154
	30	-	5	0.803	84500	-	5	0.113	12504
0.2	35	-	5	6.127	495403	-	5	1.225	105793
	40	-	5	21.879	1417253	-	5	3.997	277123
	50	-	5	1072.342	49478072	-	5	307.728	14711729
	60	6.94%	1	3043.631	97237260	1.82%	3	2290.738	74773767
	20	-	5	0.013	1947	-	5	0.004	609
	25	-	5	0.114	12616	-	5	0.040	4637
	30	-	5	0.481	39254	-	5	0.218	18706
0.3	35	-	5	3.133	204212	-	5	1.084	74413
	40	-	5	10.821	576087	-	5	4.898	266300
	50	-	5	90.217	3195695	-	5	47.728	1665147
	60	-	5	1362.553	32921438	-	5	743.674	17805962
	20	-	5	0.016	2119	-	5	0.008	1152
	25	-	5	0.088	8404	-	5	0.046	4509
	30	-	5	0.465	32787	-	5	0.220	15963
0.4	35	-	5	2.151	119618	-	5	1.210	66598
	40	-	5	4.239	181951	-	5	2.059	89991
	50	-	5	50.748	1405504	-	5	27.720	773785
	60	-	5	461.939	8411394	-	5	271.408	5005143

Table 3.9: Comparison between B&Bs and B&Bv over all weighted instances of the benchmark.

For the 140 weighted instances of Table 3.9, the revisited algorithm fails only 2 times, instead of 5, and one of the three newly solved instances avoids termination due to memory exhaustion. The two instances still unsolved improve both the upper and the lower bound, getting an optimality gap three times smaller. Computational time and branching nodes reduce by about 50% for the instances solved to optimality.

		B&Bs				B&Bv			
δ	$\|V\|$	gap	solved	CPU	BN	gap	solved	CPU	BN
0.1	20	-	5	0.020	6034	-	5	0.022	5964
	25	-	5	0.042	10326	-	5	0.048	11280
	30	-	5	0.189	32846	-	5	0.199	34788
	35	-	5	3.943	551247	-	5	3.262	410217
	40	-	5	107.544	12642269	-	5	44.817	4778665
	50	57.91%	0	OM	OM	1.25%	4	2165.374	152910911
	60	142.42%	0	OM	OM	68.40%	0	OM	OM
0.2	20	-	5	0.034	6582	-	5	0.014	2751
	25	-	5	0.434	57475	-	5	0.139	18431
	30	-	5	1.801	189567	-	5	0.614	67147
	35	-	5	20.079	1759531	-	5	11.266	1040831
	40	-	5	55.688	3775550	-	5	47.749	3390965
	50	-	5	2500.817	124134933	-	5	1974.602	100802247
	60	23.67%	0	3252.847*	122036862*	33.90%	0	OM	OM
0.3	20	-	5	0.016	2378	-	5	0.010	1604
	25	-	5	0.193	22433	-	5	0.114	13614
	30	-	5	1.069	90049	-	5	0.622	54952
	35	-	5	6.962	488122	-	5	4.799	348552
	40	-	5	29.723	1612367	-	5	14.626	837818
	50	-	5	873.641	34062285	-	5	480.459	18963773
	60	9.60%	0	3600.000	89662194	4.29%	0	3600.000	94802520
0.4	20	-	5	0.020	2655	-	5	0.013	1823
	25	-	5	0.166	16999	-	5	0.110	12119
	30	-	5	0.820	60469	-	5	0.388	30495
	35	-	5	5.449	331489	-	5	2.933	180731
	40	-	5	17.279	788391	-	5	5.800	270571
	50	-	5	214.247	6238868	-	5	56.295	1689201
	60	0.69%	4	2734.279	53504023	-	5	710.371	13912611

Table 3.10: Comparison between B&Bs and B&Bv over all weighted instances of the benchmark.

In the unweighted case (see Table 3.10), the unsolved instances decrease from 21 to 16 out of 140, with out-of-memory terminations reduced from 11 to 10. The average gaps of the unsolved instances are less than half the original ones, except for the sub-class with $\delta = 0.2$, which exhibits a moderate increase. For these instances, in fact, the weaker lower bound does not introduce enough advantages, and the consumption of memory increases. Computational times and branching nodes also improve, in particular for the denser instances, but less than in the weighted case. The larger and sparser instances ($|V| = 60$ and $\delta \in \{0.1, 0.2\}$) are difficult to compare under this respect because the computation terminates prematurely.

Chapter 4

Heuristic Methods for the WSSP

In this chapter, we introduce a number of new metaheuristic approaches to the WSSP. *The first two approaches are* Greedy Randomised Adaptive Search Procedures *with two phases: the first constructs a solution incrementally, starting from an empty set and expanding until a feasible solution is achieved; the second is a destructive procedure that refines this solution. Other three approaches improve the exploration of the solution space by extending the constructive process after the first feasible solution has been found, to visit several other ones in its surroundings. Then, we present a* Scatter Search *(*SS*) metaheuristic, that keeps in memory a multitude of suitable feasible solutions to recombine them obtaining new ones. It then refines the new discovered solutions and saves the best ones found, while discarding all the others.*

In this chapter, we present several new metaheuristic approaches to address the WSSP. The first two methods are Greedy Randomised Adaptive Search Procedures that build a solution incrementally, starting from an empty set and expanding until a feasible solution is achieved; a final destructive phase then refines this solution. Three additional approaches enhance solution space exploration by continuing the constructive process beyond the first feasible solution to explore nearby alternatives. We also introduce a Scatter Search (SS) metaheuristic, which maintains a diverse set of feasible solutions in memory, combining them to generate new candidates. These new solutions are then refined, with only the best retained and all others discarded.

As mentioned before, the WSSP is a NP-hard problem, therefore we are not always able to find an optimal solution in a reasonable time. As a matter of fact, as reported in Chapter 3, for instances with $|V| \geq 60$ the best existing approaches are not always able to find an optimal solution (or prove its optimality) in less than 1 hour. For this reason we investigate heuristic algorithms for the WSSP. The first two approaches that we present follow the framework of the *Greedy Randomised Adaptive Search Procedure* (*GRASP*) (Feo and Resende, 1989). They are based on a randomised constructive mechanism that allows multiple restarts, followed by a destructive post-processing phase to improve the result. Other three approaches improve the exploration of the solution space by extending the constructive process after the first feasible solution has been found, to visit several other ones in its surroundings. As they alternate between generating redundant solutions and reducing them by removing suitable vertices, these approaches show strong similarities to the *Large Neighbourhood Search* (*LNS*) framework (Shaw, 1998). Then, we present a *Scatter Search* (*SS*) metaheuristic, based on the idea of merging two feasible solutions, which necessarily yields another feasible solution. Since the latter is redundant, we reduce it by heuristically removing vertices.

4.1 Two *GRASP* metaheuristics

GRASP is a general constructive metaheuristic proposed by Feo and Resende (1989). It is based on the idea that, if a solution is built by subsequently adding elements to an initially empty set, it can be myopic to take deterministic choices according to a greedy criterion. In fact, selecting wrong elements in the early steps can force bad selections in the later ones and ultimately fail to generate good solutions. However, the information provided by the greedy criterion is often not fully misleading, as the elements of the optimal solution are not far behind the first suggested one. The correction proposed is, therefore, to randomly select one of the best elements at each step, instead of systematically the first. This also allows to apply the constructive scheme iteratively and obtain several different solutions. Suitable numerical parameters can be used to tune the randomisation mechanism, so as to make it more greedy or more random, and correspondingly intensify or diversify the search. In general, an auxiliary exchange heuristic is applied to improve the solutions thus generated.

This constructive approach is somehow complementary to the destructive one proposed for the *WSSP* in Macambira et al. (2019). The latter is based on randomised choices, too, but iteratively removes vertices from a subset that initially coincides with the whole vertex set V. Our choice to move in the constructive direction is partly motivated by mere curiosity and partly by the idea that optimal solutions are in general of small cardinality (see also the theoretical limit on the size of solutions for the unweighted case proved in Fujita et al. (2016)). A constructive mechanism obtains such solutions in a smaller number of iterations, potentially in shorter time and with fewer possibilities of committing mistakes with respect to a destructive one. The drawback is that it requires to move through the infeasible region until a feasible solution is found, and then remove vertices to make it minimal.

Algorithm 3 provides the pseudocode of the *GRASP* metaheuristics we propose for the *WSSP*. They start from the empty set and progressively extend it: each iteration of the loop in lines $3-6$ adds an unsafe vertex v_{new}, until all safety constraints are satisfied. Procedure `Extraction()` randomly chooses the new vertex with a selection criterium discussed in the following, that favours vertices of large degree. Since the final solution can include redundant vertices, the auxiliary function `Destructive()` turns it into a minimal one, testing the removal of each vertex in non-increasing weight order, and performing it when the result is feasible (lines $10-16$). Vertices of equal weight are considered by non-decreasing degree, following the same intuition of the construction phase.

```
input  : $G = (V, E)$: connected undirected graph
         $w : V \to \mathbb{Q}^+$: weight function on the vertices
         $\mu \in [0,1]$, $\alpha \in [0,+\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters
output : $S$: solution returned by the algorithm

Algorithm GRASP($G$, $w$, $\mu$, $\alpha$, $type$):
    $S := \emptyset$
    while $S$ is not a safe set do
        $v_{new} :=$ Extraction($G$, $S$, $V \setminus S$, $\mu$, $\alpha$, $type$)
        $S := S \cup \{v_{new}\}$
    end
    return Destructive($G, w, S$)

Function Destructive($G$, $w$, $S$):
    $Q := S$
    while $Q \neq \emptyset$ do
        $v_{max} := \arg\max_{v \in Q} w(v)$    // minimise the degree to break ties
        $Q := Q \setminus \{v_{max}\}$
        if $S \setminus \{v_{max}\}$ is a safe set then
            $S := S \setminus \{v_{max}\}$
        end
    end
    return $S$
```

Algorithm 3: The Greedy Randomised Adaptive Search Procedures

4.1.1 Selection criterium

The choice of the new vertex to extend the current subset S during the constructive phase of the algorithm is not trivial. The vertex weight seems a natural selection criterium, but it affects feasibility and optimality in opposite ways: on the one hand, vertices with a high weight allow to satisfy the safety constraints as soon as possible; on the other hand, vertices with a low weight allow to minimise the value of the objective function. Therefore, it is very hard to predict in advance the most advantageous direction to pursue at each single step.

The vertex degree plays a less obvious, but clearer role. Relaxing the definition stated in the previous chapters, we denote the *safe components* as the elements of $\mathscr{C}_G(S)$ and the *unsafe components* those of $\mathscr{C}_G(V \setminus S)$, even if S is not yet a feasible solution. Adding to S vertices of large degree is likely to split the unsafe components, and therefore to help satisfy the safety constraints. The randomised destructive heuristic of Macambira et al. (2019) removes vertices of small degree that are adjacent to unsafe vertices of small weight, to avoid building unsafe components of large weight. Our algorithm focuses on the aim to fragment the unsafe components: it neglects the weights of the adjacent vertices and replaces the overall degree with the number of adjacent *unsafe* vertices.

Definition 2. *Given a connected undirected graph $G = (V, E)$ and a subset $S \subseteq V$, let the **unsafe degree** of a vertex $v \in V$ be the number of adjacent vertices belonging to $V \setminus S$.*

$$\delta^{V \setminus S}(v) = |\{u \in V \setminus S \mid (u, v) \in E\}|$$

Notice that at the beginning, when $S = \emptyset$, the unsafe degree of any vertex coincides with its degree.

4.1.2 Randomisation schemes

Algorithm 4 reports the pseudocode of the `Extraction()` procedure, that selects the new vertex to add to S applying one of two alternative randomisation mechanisms. Notice that the candidate set C from which the vertex is extracted coincides with $V \setminus S$ in the *GRASP* metaheuristics, but it will be reduced in the following algorithms.

```
input : $G = (V, E)$: connected undirected graph
        $S$: current set of safe vertices
        $C$: candidate set of unsafe vertices
        $\mu \in [0,1], \alpha \in [0, +\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters
output: $v_{new}$: vertex selected to be added to $S$

Function Extraction($G, S, C, \mu, \alpha, type$):
    forall $v \in C$ do $\delta^{V \setminus S}(v) = |\{u \in V \setminus S \mid (u, v) \in E\}|$
    if $type = RCL$ then
        $\delta_{\min} := \min_{v \in C} \delta^{V \setminus S}(v)$
        $\delta_{\max} := \max_{v \in C} \delta^{V \setminus S}(v)$
        $\delta_{\mu} := \mu \cdot \delta_{min} + (1 - \mu) \cdot \delta_{max}$
        $C' := \{v \in C \mid \delta^{V \setminus S}(v) \geq \delta_{\mu}\}$
        forall $v \in C'$ do $\pi(v) := 1/|C'|$
        forall $v \in C \setminus C'$ do $\pi(v) := 0$
    else if $type = HBSS$ then
        forall $v \in C$ do $\varphi(v) := (\delta^{V \setminus S}(v))^{\alpha} + 1$
        forall $v \in C$ do $\pi(v) := \varphi(v) / \sum_{x \in C} \varphi(x)$
    end
    $v_{new} \leftarrow$ random extraction from $C$ with probability $\pi(\cdot)$
    return $v_{new}$
```

Algorithm 4: The randomised selection procedure

The *Restricted Candidate List* (*RCL*) mechanism (lines 4 – 9) determines the minimum and the maximum unsafe degree of the unsafe vertices, respectively denoted as $\delta_{\min}$ and $\delta_{\max}$. Then, it computes a convex combination δ_{μ} of these values with a suitable parameter $\mu \in [0, 1]$. Such a combination is used as a threshold, to build a set C' of candidate vertices with large unsafe degree. Then, a candidate vertex is selected at random with a uniform probability. Large values of μ correspond to more random choices, small values to more greedy choices. Even when $\mu = 0$, however, the mechanism is not deterministic, because several vertices can have the maximum unsafe degree.

The *Heuristic Biased Stochastic Sampling* (*HBSS*) mechanism (lines 11 – 12) considers all candidate vertices, but assigns them different probabilities, so as to bias the selection in favour of those with larger unsafe degree. In particular, the probability of each candidate vertex v is proportional to $(\delta^{V\setminus S}(v))^\alpha + 1$, where the unsafe degree is raised to a suitable power $\alpha \geq 0$ and increased by 1 to guarantee a non-zero probability of being selected also to vertices with no unsafe adjacent vertices. The smaller is α, the more random the choice; the larger it is, the greedier.

In summary, the two *GRASP* metaheuristics combine a randomised constructive phase, based on the vertex unsafe degree and focused on feasibility, and a deterministic destructive phase, based on the weight of the vertices and focused on optimality. The latter can be interpreted also as a local search improvement method. In fact, consider the neighbourhood that includes the feasible solutions obtained adding or removing a single vertex. A steepest descent algorithm based on it would behave as the destructive procedure: it would always select the vertex of maximum weight that can be feasibly removed from the solution. Larger neighbourhoods could be more effective, but the combination of weight and connectivity aspects of the safety constraints would probably make their exploration quite inefficient.

The complexity of the *GRASP* metaheuristics can be easily computed based on the pseudocode of Algorithms 3 and 4. The constructive phase calls for at most $|V|$ times the `Extraction()` procedure, whose complexity is $T_{\text{Extr}} \in O(|E| + |V|)$, because it scans the $|E|$ edges of the graph to compute $\delta^{V\setminus S}$ and the $|V|$ vertices to determine their probabilities and select one. The destructive phase has complexity $T_{\text{Destr}} \in O(|V|(|E| + |V|))$, because it determines which of the $O(|V|)$ vertices of S can be feasibly removed by visiting the induced sub-graph in $O(|E| + |V|)$ time. Overall, the computational time is $T_{\text{GRASP}} \in O(|V|(|E| + |V|))$.

4.2 A heuristic with Delayed Termination

Our computational experience suggests that halting the constructive phase right after the achievement of feasibility tends to bind too much the following destructive phase. Introducing redundant vertices creates more expensive solutions, but these often contain better minimal solutions, that the destructive procedure is able to identify. In other words, starting from the whole vertex set is probably excessively loose, but starting from the first feasible solution found can be too restrictive. We investigate two complementary ways of delaying the termination of the algorithm introducing additional vertices. The former, denoted as *Simple Delayed Termination* (*SDT*), moves from a feasible solution further into the feasible region. The latter, denoted as *Avoidant Delayed Termination* (*ADT*) prefers the insertion of vertices that are promising, but keep the solution infeasible, in order to explore the border of the feasible region. As will be clear from the analysis of their computational complexity, both approaches take more time per iteration than the *GRASP* approaches described above. However, they both generate several feasible solutions, instead of a single one, and therefore possibly provide better results.

4.2.1 Simple Delayed Termination

The pseudocode of the *SDT* heuristic is reported in Algorithm 5. It starts from the feasible solution generated by the *GRASP* metaheuristic (lines 2 – 7). Then, it further enlarges the solution, performing a number $\gamma|V|$ (with $\gamma \in [0,1]$) of extra iterations proportional to the number of vertices. When $\gamma = 0$, the *SDT* heuristic reduces to the *GRASP* algorithm.

```
input  : $G = (V, E)$: connected undirected graph
         $w : V \to \mathbb{Q}^+$: weight function on the vertices
         $\mu \in [0,1], \alpha \in [0,+\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters
         $\gamma \in [0,1]$: delay factor
output : $S^*$: solution returned by the algorithm

1  Algorithm SDT($G, w, \mu, \alpha, type, \gamma$):
2      $S := \emptyset$
3      while $S$ is not a safe set do
4          $v_{new}$ := Extraction($G, S, V \setminus S, \mu, \alpha, type$)
5          $S := S \cup \{v_{new}\}$
6      end
7      $S^*$ := Destructive($G, w, S$)
8      for $l = 1, \dots, \gamma|V|$ do
9          $C := \{v \in V \setminus S \mid \exists u \in S : (u,v) \in E\}$
10         $v_{new} := \arg\min_{v \in C} w_v$        // maximise the degree to break ties
11         $S := S \cup \{v_{new}\}$
12         $S'$ := Destructive($G, w, S$)
13         if $w(S') < w(S^*)$ then $S^* := S'$
14     end
15     return $S^*$
```

Algorithm 5: The Simple Delayed Termination heuristic

Differently from the first constructive phase, the additional one (lines 8 – 14) starts from a feasible solution and extracts the new vertex only from a restricted candidate set C composed by the unsafe vertices adjacent to S, so that feasibility is strictly preserved:

$$C := \{v \in V \setminus S \mid \exists u \in S : (u,v) \in E\}$$

Since splitting the unsafe components to pursue feasibility is no longer necessary, the selection mechanism is not mainly based on the degree: it simply chooses the candidate vertex of minimum weight (using the maximum degree as a tie breaker), in order to worsen the objective function as little as possible.

On each enlarged solution S the algorithm applies the destructive procedure to obtain an improved one, S', and possibly update the best known one, S^*. Notice that the following extra iterations will add new vertices to the redundant solution S, not to the minimal solution S'. The rationale of the algorithm, in fact, is that the redundant vertices introduced could help the destructive procedure get rid of vertices misleadingly added by the first randomised constructive phase. Of course, the destructive phase could remove the extra vertices, and possibly regenerate already known solutions, but the extra vertices have been chosen minimising the weight. So, the destructive phase will consider them later, and remove them only if necessary.

The extra constructive phase and the final destructive phase of the *SDT* heuristic can be interpreted also as a *LNS* procedure (Shaw, 1998) that augments a feasible solution and makes it minimal again, adding and removing redundant elements with a heuristic criterium.

The complexity of the *SDT* heuristic is given by the starting *GRASP* procedure of lines 2 – 7, followed by $\gamma|V|$ iterations of the loop at lines 9 – 13, whose most expensive subroutine is the `Destructive()` procedure. The resulting complexity is $T_{\text{SDT}} \in O(T_{\text{GRASP}} + |V| \cdot T_{\text{Destr}}) = O(|V|^2(|E| + |V|))$.

4.2.2 Avoidant Delayed Termination

The *ADT* heuristic is complementary to the *SDT* heuristic. Its basic idea is to replace the simple random choice of a new vertex with a full exploration of all available choices, possibly finding several feasible solutions and improving each one with the destructive procedure. The termination of the algorithm is delayed after finding the first feasible solution with the aim to remain in the infeasible region, but still include promising vertices (with large unsafe degree). The purpose of this choice is to try and obtain that the current infeasible subset is close (in terms of Hamming distance) to many feasible solutions that provide good starting points for the destructive procedure. Another possible interpretation of the *ADT* heuristic is that it combines the *GRASP* constructive mechanism with an exploration of the neighbourhood obtained replacing the last added vertex with another one. This is clearly an intensification mechanism, to better investigate the region of the solution space chosen by the randomised constructive mechanism.

Algorithm 6 provides a detailed pseudocode. The *ADT* heuristic starts from the empty set and initialises the best known solution as the whole vertex set. At each iteration, it performs an exploration phase that considers all unsafe vertices as candidates and tests for each of them whether adding it to S yields a feasible solution. If it does, the solution is improved by the destructive procedure and compared with the best known one, in order to keep it updated, and the vertex is removed from the candidates. After the exploration, one of the remaining candidate vertices is extracted at random, with the randomisation schemes already described in Section 4.1.2, and added to the current solution. If all extensions of the current set are feasible, the candidate set is empty and the algorithm terminates. Otherwise, the termination is delayed, as in the *SDT* heuristic, until the number of extra vertices added after finding the first feasible solution exceeds $\gamma|V|$. Notice that, even when $\gamma = 0$, the algorithm does not reduce to the *GRASP* metaheuristic, because the additional exploration phase in general finds several feasible solutions, instead of a single one.

The single steps of the *ADT* heuristic are also much slower. In fact, each of the $O(|V|)$ iterations in the loop of lines 6 – 22 applies the loop of lines 8 – 15 to the $O(|V|)$ candidate vertices of set C. Each iteration of this inner loop tests the feasibility of the current solution at line 9 with a visit in $O(|V|+|E|)$ time. In the positive case, it applies the `Destructive()` procedure, that takes T_{Destr}. Out of the inner loop (lines 17 – 18), a new vertex is selected in time T_{Extr} and added to the current solution. In summary, the complexity is $T_{ADT} \in O(|V|[|V|(|V|+|E|+T_{\text{Destr}})+T_{\text{Extr}}]) = O(|V|^3(|E|+|V|))$. This is clearly larger than *SDT*, though it should be remarked that the `Destructive()` procedure is not applied at every iteration and on the whole graph.

```
input  : $G = (V, E)$: connected undirected graph
         $w : V \to \mathbb{Q}^+$: weight function on the vertices
         $\mu \in [0,1], \alpha \in [0,+\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters
         $\gamma \in [0,1]$: delay factor
output : $S^*$: solution returned by the algorithm

1  Algorithm ADT($G$, $w$, $\mu$, $\alpha$, $type$, $\gamma$):
2      $S := \emptyset$
3      $S^* := V$
4      $l := 0$
5      found := false
6      repeat
7          $C := V \setminus S$
8          for $c \in V \setminus S$ do
9              if $S \cup \{c\}$ is a safe set then
10                 found := true
11                 $S' :=$ Destructive($G, w, S \cup \{c\}$)
12                 if $w(S') < w(S^*)$ then $S^* := S'$
13                 $C := C \setminus \{c\}$
14             end
15         end
16         if $C = \emptyset$ then break
17         $v_{new} :=$ Extraction($G$, $S$, $C$, $\mu$, $\alpha$, $type$)
18         $S := S \cup \{v_{new}\}$
19         if found then
20             $l := l + 1$
21         end
22     until $l > \gamma |V|$
23     return $S^*$
```

Algorithm 6: The Avoidant Delayed Termination heuristic

4.2.3 Truncated Avoidant Delayed Termination

The exploration phase that evaluates all candidate vertices yields potential improvements, but also implies a significant increase in complexity. The *Truncated Avoidant Delayed Termination* (*TADT*) heuristic reduces the exploration effort to achieve a compromise between efficiency and effectiveness. The idea is to randomly extract the candidate vertices and stop as soon as one generates an infeasible solution, instead of considering all of them. Notice that the random extraction favours the candidates that are more likely to yield feasible solutions, if possible.

```
input  : $G = (V, E)$: connected undirected graph
         $w : V \to \mathbb{Q}^+$: weight function on the vertices
         $\mu \in [0,1], \alpha \in [0,+\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters
         $\gamma \in [0,1]$: delay factor
output : $S^*$: solution returned by the algorithm

Algorithm TADT($G, w, \mu, \alpha, type, \gamma$):
    $S := \emptyset$
    $S^* := V$
    $l := 0$
    found := false
    repeat
        $C := V \setminus S$
        stop := false
        while not stop and $C \neq \emptyset$ do
            $v_{new}$ := Extraction($G, S, C, \mu, \alpha, type$)
            $C := C \setminus \{v_{new}\}$
            if $S \cup \{v_{new}\}$ is a safe set then
                found := true
                $S'$ := Destructive($G, w, S \cup \{v_{new}\}$)
                if $w(S') < w(S^*)$ then $S^* := S'$
            else
                $S := S \cup \{v_{new}\}$
                if found then
                    $l := l + 1$
                end
                stop := true
            end
        end
    until $l > \gamma|V|$ or $C = \emptyset$
    return $S^*$
```

Algorithm 7: The Truncated Avoidant Delayed Termination heuristic

Algorithm 7 provides the pseudocode. The *TADT* heuristic starts from an empty set, initialises the best known solution with the whole vertex set and considers all unsafe vertices as candidates. Instead of scanning them systematically, it extracts one candidate at random with the already described randomisation schemes, and checks whether adding it to the current solution makes it feasible. If it does, the algorithm improves the solution, possibly updates the best known one and proceeds to another extraction, removing the used vertex from the candidate set (and, of course, not adding it to the solution), as in the regular *ADT*. If the solution obtained is not feasible, however, the algorithm straightly proceeds with the next iteration, ignoring all candidates left. This reduces the computational effort, while possibly missing feasible solutions. The rationale is that, since the extraction is biased in favour of the more promising vertices, the unexplored solutions are likely to be also infeasible, or at least less interesting than the explored ones. This approach is similar in principle to the so called *first-best* exploration strategy of large neighbourhoods, that accepts the first improving neighbour solution found, as opposed to the *global-best* strategy, that selects the best one (Hansen and Mladenović, 2006).

When $\gamma = 0$, the *TADT* heuristic explores the same infeasible solutions as *GRASP* (if the pseudorandom numbers are the same), which form a subset of those explored by *ADT*. The difference between *GRASP* and *TADT*, with $\gamma = 0$, is that the former terminates as soon as it finds a feasible solution, whereas the latter explores other feasible solutions, terminating only at the first infeasible one.

The complexity of the *TADT* heuristic is the same as *ADT* in the worst case, that corresponds to finding all feasible solutions at each iteration of the loop in lines 9 – 23, and therefore entering the block in lines 13 – 15, before the unfeasible solution that terminates the loop. The resulting complexity is, consequently, $T_{\text{TADT}} \in O(|V|^3(|E| + |V|))$, but the worst case is likely to be less frequent than in *ADT*.

4.3 A Scatter Search metaheuristic

In this section we discuss a new *SS* metaheuristic for the *WSSP*. Scatter Search is a general-purpose approach to optimisation introduced by Glover (1977). It requires an auxiliary basic algorithm to generate a *reference set*, composed by feasible solutions suitably selected among the ones found during the computation. The reference set is divided into two disjoint subsets B and D: the former includes the best n_B solutions found so far, the latter the most “diverse” n_D solutions. The concept of diversity, defined in detail in the following, derives from the Hamming distance between the incidence vectors of the solutions. The main idea of *SS* is to “recombine” pairs of solutions drawn from the reference set so as to obtain new solutions, that become candidates to join the reference set.

Algorithm 8 reports our adaptation of this basic scheme to the *WSSP*.

```
Input  : $G = (V, E)$: connected undirected graph
         $w : V \to \mathbb{R}^+$: weight function on the vertices
         $\mu \in [0,1]$, $n_B, n_D \in \mathbb{N}$: parameters
Algorithm Scatter Search($G, w, \mu, n_B, n_D$):
    $B = \emptyset$ // best known reference solutions
    $D = \emptyset$ // diverse reference solutions
    while $|D| < n_B + n_D$ do // Initialisation of the reference set
        $S =$ GRASP($G, w, \mu$)
        if $S \notin D$ then $D = D \cup \{S\}$
    end
    for $i = 1, \ldots, n_B$ do // Selection of the best reference solutions
        $S = \arg\min_{X \in D} w(X)$
        $D = D \setminus \{S\}$
        $B = B \cup \{S\}$
    end
    for $X \in D$ do // Computation of the diversity indices
        $\delta[X] = \sum_{Y \in B \cup D \setminus \{Y\}}$ Hamming($X, Y$)
    end
    change = true
    while change do // As long as the reference set changes
        change = false
        for $S_1 \in B$ do
            for $S_2 \in B \cup D : S_1 \neq S_2$ do
                $S =$ Destructive($G, w, S_1 \cup S_2$) // Merge and improve $S_1$ and $S_2$
                $Y = \arg\max_{X \in B} w(X)$ // Find the candidate to leave $B$
                if $w(S) < w(Y) \wedge S \notin B$ then
                    $B = B \cup \{S\} \setminus \{Y\}$ // Replace $Y$ with $S$
                    for $X \in D$ do $\delta[X] = \delta[X]+$Hamming($X, S$)$-$Hamming($X, Y$)
                    $S = Y$ // The new candidate to enter $D$ is $Y$
                    change = true
                end
                $Y = \arg\min_{X \in D} \delta[X]$ // Find the candidate to leave $D$
                $\delta[S] = \sum_{X \in B \cup D \setminus \{Y\}}$ Hamming($S, X$)
                if $\delta[S] > \delta[Y] \wedge S \notin D$ then
                    $D = D \cup \{S\} \setminus \{Y\}$ // Replace $Y$ with $S$
                    for $X \in D$ do $\delta[X] = \delta[X]+$Hamming($X, S$)$-$Hamming($X, Y$)
                    change = true
                end
            end
        end
    end
    return $S^* = \arg\min_{X \in B} w(X)$
```

Algorithm 8: The Scatter Search Heuristic

First we generate $n_B + n_D$ reference solutions with the *GRASP* metaheuristic described in Section 4.1, making sure that they are all different from each other. Once the reference set is generated, we move its n_B best solutions to B (the set of best known reference solutions) and leave the remaining ones in D (the set of diverse reference solutions). Afterwards, for each solution $X \in D$ we compute its diversity index $\delta[X]$ summing the Hamming distances between X and every other solution in the whole reference set $B \cup D$.

Now, the core of the algorithm iteratively tries to update the reference set: every time it succeeds, it sets the flag variable change to `true` in order to imply another iteration; when the reference set is unchanged, the process terminates. Each iteration scans pairs of non-identical reference solutions (S_1, S_2). The former must be of good quality, whereas the latter can have either a small cost or a high diversity index to pursue, respectively, intensification or diversification. We now recombine the two solutions, merging them and removing the redundant vertices with a destructive procedure. We show in the following that this guarantees to produce a feasible solution S. This solution becomes a candidate to join the reference set. First we test whether S should enter subset B by identifying the worst solution in B (denoted as Y in the pseudo-code), that is the candidate to leave B. If S is strictly better than Y and is not already in B, then we proceed to replace Y with S. Now S is in the reference set and Y is out, but it might be qualified to enter D. Therefore, we save Y in S for the following step. The flag change becomes `true` to mark the modification of the reference set and the diversity value of every solution in D is updated accordingly. Then, we compute the diversity index of S (be it the original recombined solution or the one expelled from B) and identify the solution with smallest diversity index in D as the candidate to leave that set (once again, denoted as Y). If S is strictly more diverse than Y and it is not already in D, we replace Y with S, set the flag change to **true** and update the diversity indices.

The choice of *GRASP* over the Delayed Termination Heuristics is motivated by the faster computation of the former and the fewer parameters to tune, in order to focus on the effect of the recombination mechanism.

Recombination The recombination procedure at Line 21 of Algorithm 8 exploits the following basic property of the *WSSP* to derive a new safe set from any given pair of safe sets.

Proposition 8. *Given a graph $G = (V, E)$ with vertex weights $w : V \to \mathbb{R}^+$, if S and S', are safe sets for (G, w), then $S \cup S'$ is also a safe set.*

Proof. Let $\tilde{S} \in \mathscr{C}_G(S \cup S')$ be a connected component induced by the vertices of the merged safe sets and $\tilde{U} \in \mathscr{C}_G(V \setminus (S \cup S'))$ a connected component induced by the vertices that do not belong to any of them. We will show that, if $\tilde{S}$ and $\tilde{U}$ are adjacent, they always satisfy the safety constraint, and therefore $S \cup S'$ is safe. Since $\tilde{U}$ is connected and its vertices are unsafe in S, there exists an unsafe component $U_j \in \mathscr{C}_G(V \setminus S)$ that contains it: $\tilde{U} \subseteq U_j$. As well, there exists $U'_j \in \mathscr{C}_G(V \setminus S')$ such that $\tilde{U} \subseteq U'_j$. If $\tilde{S} \bowtie \tilde{U}$, by definition there is an edge $(u, v) \in E$ with $u \in \tilde{U}$ and $v \in \tilde{S}$. Suppose, without loss of generality, that $v \in S$ and $S_i \in \mathscr{C}_G(S)$ is the connected component that contains it in the original solution S, and notice that $u \in U_j$ and $U_j \bowtie S_i$. The feasibility of S implies that $w(U_j) \leq w(S_i)$ and, therefore, the following chain of inequalities hold

$$w(\tilde{U}) \leq w(U_j) \leq w(S_i) \leq w(\tilde{S})$$

implying the thesis. $\square$

The previous proposition allows to derive a new feasible solution from any pair of elements in the reference set. Obviously, such a solution is non-minimal and, therefore, of poor quality. For this reason, every time we produce a recombined solution, we improve it removing redundant vertices with the destructive mechanism described above for the *GRASP* procedure (see Algorithm 3).

Computational complexity The initialisation phase of the *SS* algorithm requires $n_B + n_D$ calls to the GRASP heuristic, which has complexity $O(|V| \cdot (|E| + |V|))$. For the sake of simplicity, this analysis assumes that no duplicate solutions are generated, though, in very degenerate cases, the iterations to reach the required number of different solutions might be unlimitedly numerous. It also requires to classify the reference solutions into either best known or diverse ($O(n_B \log(n_B + n_D))$ time) and to compute the diversity indices, which takes $O(n_D(n_B + n_D)) \cdot |V|)$ operations. This is negligible with respect to the main loop. In fact, each iteration of the loop scans $n_B(n_B + n_D - 1)$ pairs of reference solutions. It merges them and feeds the result to the destructive procedure, whose complexity is in $O(|V| \cdot (|V| + |E|))$. The update of the reference set requires to check whether the recombined solution is different from the reference ones and qualified to enter subsets B or D. This can be done in $O((n_B + n_D)|V|)$ time, while the update of the diversity indices takes $O(n_D|V|)$. In conclusion, the total complexity of the heuristic is $O(L \cdot n_B(n_B + n_D) \cdot |V| \cdot (|V| + |E| + n_B + n_D)$, where L is the number of iterations. Typically, parameters n_B and n_D are small constants and the complexity reduces to $O(L \cdot |V| \cdot (|V| + |E|))$.

Multiple restart Algorithm 8 terminates when the reference set converges to a set that the deterministic recombination mechanism is unable to modify. In order to let the algorithm proceed beyond convergence, we can simply restart it: the *GRASP* procedure used in the initialisation step, in fact, is usually able to generate new solutions through randomisation. This will allow to perform experiments with a given time limit in order to compare the *SS* metaheuristic to the state of the art approaches. Of course, the time limit also imposes a premature termination condition within Algorithm 8 when it is reached before convergence.

4.4 Computational experiments

This section describes the computational results obtained by the algorithms presented above on the benchmarks of large instances.

At first we present the tuning of the parameters for the *GRASP* metaheuristics (Section 4.4.1), the delayed termination algorithms (Section 4.4.2) and the Scatter Search (Section 4.4.3). Unexpected results on the benchmark $\mathtt{H}^+$, suggest that these instances are particularly easy to solve. We will also propose a conjecture regarding this behaviour, which aligns with certain theoretical results from the literature. Then, we will compare the performance of the constructive-destructive algorithms with each other (Section 4.4.4) and the performance of the best one with the only existing heuristic in the literature and with the known optimal values (Section 4.4.5). Once again, we compare the Scatter Search with the best heuristic that arises from the previous experiments (Section 4.4.6). Additionally, we will discuss the structure of the solutions, focusing in particular on the number of safe and unsafe components (Section 4.4.7). At last, we present the results obtained by the branch-and-bound algorithm presented in Chapter 3 (flexible version) with a heuristic initialisation.

Definition of the optimality gap To measure the quality of the solution returned by a heuristic algorithm, usually one computes the optimality gap

$$(UB - z^*)/z^*$$

where, as in Section 3.5 UB is a valid upper bound and z^* is the value of the optimal solution. Unfortunately, optimal results are available only for instances up to 60 vertices. One could replace z^* with a lower bound but, for the considered instances the quality of such values is quite bad. Consequently, the results will be evaluated computing the percent gap

$$(UB - UB^*)/UB^* \tag{20}$$

where UB^* is the value of the best known solution for the given instance. This underestimates the actual optimality gap and the dissimilarities between different parameter tunings and algorithms, but is the best possible measure with the available information.

4.4.1 Parameter tuning for the *GRASP* metaheuristics

Both *GRASP* metaheuristics have a single parameter that tunes the balance between a greedy and a random behaviour. The *RCL*-based heuristic depends on a real parameter $\mu \in [0,1]$, for which we consider seven possible values ($\mu \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$) that progressively enlarge the restricted candidate set, making more random choices. The *HBSS*-based heuristic depends on parameter $\alpha \geq 0$, for which we consider four possible values: $\alpha \in \{0,1,2,3\}$, that progressively bias the choice in favour of the vertices with a high unsafe degree, making it less random.

We have first applied the 11 resulting heuristics (one for each parameter value considered: 7 for *RCL* and 4 for *HBSS*) with a time limit of 10 seconds on the large random graphs of benchmark $\mathtt{H}^+$. The outcome was completely unexpected: on a large majority of instances (140 out of 200) all heuristics returned the same result. What is more, this result is the best one found by all algorithms, including also the large number of *SDT*, *ADT*, *TADT* and *SS* variants discussed in the following. The phenomenon shows a clear trend with respect to the size, density and weight distribution of the instances. Table 4.1 reports for each weight function (on the left the weighted instances, on the right the unweighted ones), size (on the rows) and density (on the columns) the maximum gap obtained by any of the 11 *GRASP* heuristics on any of the 5 instances with the given features. Label "-" marks the classes in which all settings obtained the best result for all instances: they are all characterised by a large size $|V|$ and density δ. The convergence of these values to zero is sharp, and quicker for the unweighted instances than for the weighted ones.

	weighted				unweighted			
$\|V\|$	0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4
100	26.2%	5.8%	2.0%	1.2%	14.3%	2.0%	-	-
150	6.6%	1.2%	-	-	2.8%	-	-	-
200	2.9%	-	-	-	1.0%	-	-	-
250	1.2%	-	-	-	-	-	-	-
300	0.6%	-	-	-	-	-	-	-

Table 4.1: Maximum gap of the solutions returned by the 11 different parameter settings of the *GRASP* metaheuristics on each group of 5 instances of benchmark $\mathtt{H}^+$ with the same number of vertices, density and weight distribution

Our conjecture is that these results are optimal, but we have no formal proof for this statement. Hypothetically, an unknown flaw could affect all algorithms and induce them to generate the same result, but only if the instance is sufficiently large and dense. What makes this explanation unlikely is that the solution systematically returned has nearly always the same structure: the safe set forms a single component with a value equal to half the total weight of the graph. These results are consistent with Cordone and Franchi (2023), where the authors demonstrate the asymptotic behaviour of random graphs, indicating that the optimal solution of the safe set tends to partition the graph into a single safe component and a single unsafe component of nearly equal weight.

Table 4.2 reports the average gaps achieved by the *GRASP* metaheuristics with their 11 different tunings on the large benchmarks. The upper half refers to the weighted instances and the lower half to the unweighted ones. Each row concerns a class of instances (plus a row with the overall average), each column a different tuning of the randomisation parameter, with the *RCL* mechanism on the left and the *HBSS* one on the right. We provide aggregate results because no interesting dependency appears with respect to the size or density of the instances.

		μ							α			
	class	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0	1	2	3
weighted	`H`$^+$	1.12%	0.99%	**0.93%**	**0.93%**	1.01%	1.18%	1.31%	1.68%	1.68%	1.62%	1.66%
	`SW`	34.86%	33.99%	28.09%	29.17%	**27.83%**	30.90%	31.29%	29.01%	29.09%	30.22%	30.37%
	`Reg`	**7.22%**	7.36%	8.21%	8.90%	8.88%	9.14%	11.02%	11.54%	11.54%	11.60%	12.27%
	`Pla`	12.53%	12.36%	10.12%	**9.23%**	10.26%	9.70%	13.26%	14.71%	14.76%	16.30%	16.76%
	`Grid`	7.30%	6.97%	**5.71%**	7.05%	6.98%	6.99%	8.69%	9.58%	9.58%	8.54%	7.78%
	`all`	11.07%	10.83%	**9.44%**	9.78%	9.63%	10.35%	11.34%	11.32%	11.34%	11.68%	11.89%
unweighted	`H`$^+$	0.46%	0.38%	**0.25%**	0.39%	0.31%	0.45%	0.67%	0.74%	0.74%	0.74%	0.75%
	`SW`	18.70%	18.76%	**16.65%**	16.82%	17.28%	18.07%	21.54%	23.41%	23.41%	22.48%	21.88%
	`Reg`	**2.82%**	**2.82%**	3.01%	3.17%	3.66%	3.71%	4.11%	5.27%	5.29%	5.11%	5.03%
	`Pla`	9.48%	8.35%	7.03%	**6.14%**	6.34%	6.73%	7.60%	8.15%	8.15%	7.08%	7.41%
	`Grid`	51.09%	50.67%	36.18%	26.53%	17.97%	18.91%	**10.62%**	12.39%	12.39%	12.55%	13.27%
	`all`	7.96%	7.80%	6.58%	6.20%	**6.03%**	6.34%	7.00%	7.81%	7.82%	7.47%	7.40%

Table 4.2: Average gaps produced by the different versions of the GRASP metaheuristic for each class of instances

As already observed, the random instances of benchmark `H`$^+$ are easy, in particular in the unweighted case. On the contrary, the other benchmarks exhibit larger gaps, that vary strongly depending on the instance class and on the parameter tuning. Unfortunately, no tuning is consistently superior for all classes. For the weighted instances, the *RCL* mechanism with $\mu = 0.2$ is the best on average and reasonably good for all classes. The Wilcoxon's signed rank test (Wilcoxon, 1945) provides significant p-values for the better performance of this tuning over all *HBSS* variants ($p < 10^{-6}$) and over the *RCL* variants with $\mu = 0.0$, 0.5 and 0.6 ($p < 1\%$), but not for the other values. For the unweighted instances, the *RCL* mechanism with $\mu = 0.4$ is the best on average and reasonably good for all classes except for the grid instances. Wilcoxon's test, however, supports this choice with significant p-values only with respect to the *HBSS* variants ($p < 10^{-6}$) and the *RCL* variants with $\mu = 0.6$ ($p < 0.0002$), while the other comparisons are non-significant. Notice that, when making a series of hypothesis tests, the probability of coming to at least one false conclusion by chance, known as *family-wise error rate*, should be kept under control. This can be done, in a conservative way, by applying the Bonferroni correction (Dunn, 1961), that amounts to dividing the required threshold by the number of tests performed. Once this correction is applied, the only statistical results that remain significant are that the chosen variants ($\mu = 0.2$ for the weighted instances and $\mu = 0.4$ for the unweighted ones) perform better than all the HBSS variants.

4.4.2 Parameter tuning of the Delayed Termination approaches

The delayed termination heuristics depend on two parameters: one rules the randomisation mechanism also used in the *GRASP* approaches, the other is the delay factor γ that determines how many additional vertices to introduce in the solution after finding the first feasible solution.

Even if the *RCL* mechanism dominates the *HBSS* heuristic, we decided to keep both of them, and all of the parameter tunings previously considered. This is partly due to the limited statistical significance of the results obtained on *GRASP*, and partly on the idea that the delayed termination could interact in hardly predictable ways with the randomisation scheme. The best tuning of randomisation with the standard termination could easily be very different from the best tuning with a delayed termination. Randomisation, in fact, is a classical diversification mechanism, whereas the delayed termination is an intensification mechanism: they play complementary roles.

We therefore consider the same values of μ and α used above, while γ is set in $\{0.0, 0.1, 0.2, 0.3, 0.4\}$. Tables 4.3, 4.4 and 4.5 report the overall average gaps with respect to the best known result on the large benchmarks `H+`, `SW`, `Reg`, `Pla` and `Grid`, respectively for the *SDT*, *ADT* and *TADT* heuristics. The upper half of the tables is dedicated to the weighted instances, the lower half to the unweighted ones. The left part considers the 7 possible values of parameter μ for the *RCL* mechanism, the right part the 4 values of α for the *HBSS* mechanism. Each row refers to one of the 5 values of the delayed termination parameter γ.

Concerning *SDT*, the best average gap for the weighted instances is 2.84% (marked in bold), and is obtained by $\mu = 0.3$ and $\gamma = 0.2$, with a maximum gap equal to 21.55% on the whole benchmark. Wilcoxon's test suggests that there is a subset of parameter settings for which the difference in performance with respect to the best one is not statistically significant. We have shaded in dark grey the settings for which the p-value of these tests is ≥ 0.05 and in light grey those for which $0.05 > p \geq 10^{-4}$. This reduction of the p-value by a factor of 500 aims to account for the family-wise error rate induced by the large number of tests performed, applying the (rather conservative) Bonferroni correction.

For the unweighted instances, unexpectedly, the *HBSS* mechanism with $\alpha = 0$ or 1 and $\gamma = 0.1$ obtains a 4.01% average gap, with a maximum of 27.91%. The two tunings have very similar results, probably due to the fact that the weights of all vertices and the degrees of many are the same, which makes the selection of promising elements a hard task for the algorithm. The *RCL* mechanism tends to perform worse than the *HBSS* one on average, contrary to what happened in the *GRASP* algorithms. However, in this case, Wilcoxon's test suggests a wide subset of settings that are not statistically dominated, and most of these settings adopt the *RCL* mechanism, including the ones that performed well on the weighted instances.

SDT		μ							α			
	γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0	1	2	3
weighted	0.0	11.15%	10.88%	9.46%	9.86%	9.76%	10.45%	11.38%	11.38%	11.37%	11.70%	11.98%
	0.1	4.50%	4.21%	3.85%	3.72%	3.93%	4.47%	5.20%	6.05%	6.03%	5.90%	5.39%
	0.2	3.62%	3.55%	3.17%	**2.84%**	3.03%	3.46%	3.84%	4.38%	4.39%	4.30%	4.49%
	0.3	4.06%	3.86%	3.26%	3.35%	3.39%	3.75%	4.41%	4.65%	4.66%	4.54%	4.55%
	0.4	4.55%	4.37%	4.03%	3.77%	4.07%	4.35%	4.88%	5.19%	5.14%	5.07%	5.00%
unweighted	0.0	8.03%	7.85%	6.56%	6.26%	6.09%	6.43%	6.89%	7.82%	7.82%	7.57%	7.55%
	0.1	6.05%	6.00%	4.57%	4.37%	4.27%	4.22%	4.14%	**4.01%**	**4.01%**	4.17%	4.61%
	0.2	5.62%	5.85%	5.32%	4.71%	4.71%	4.69%	4.78%	4.65%	4.65%	5.16%	5.16%
	0.3	6.13%	6.15%	5.47%	5.39%	5.49%	5.18%	5.24%	5.41%	5.40%	5.83%	5.84%
	0.4	6.51%	6.58%	5.86%	5.94%	6.01%	5.62%	5.70%	5.97%	5.97%	6.16%	6.12%

Table 4.3: Average gaps produced by the different versions of the *SDT* heuristic on the large instances of benchmarks H$^+$, `SW`, `Reg`, `Pla` and `Grid`. In dark grey the settings for which the p-value is ≥ 0.05 and in light grey those for which $0.05 > p \geq 10^{-4}$.

Concerning *ADT* (Table 4.4), the best average gap, that is 9.29%, is obtained by $\mu = 0.2$ and $\gamma = 0.4$, and the maximum one is equal to 49.11%. The unweighted instances behave in a similar way, with the best tuning in $\mu = 0.3$ and $\gamma = 0.4$, yielding an average gap equal to 7.82% and a maximum one equal to 52.94%. These gaps are much larger than those obtained by *SDT*. This could suggest that keeping on the border of the feasible region is a less effective strategy than diving into it, but it could also simply be due to the stronger computational cost of *ADT*. The *RCL* mechanism tends to be better than *HBSS*: nearly all good parameter settings adopt the former.

ADT		μ							α			
	γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0	1	2	3
weighted	0.0	19.91%	19.99%	18.78%	18.77%	19.47%	19.70%	22.17%	22.38%	22.36%	23.25%	22.38%
	0.1	12.84%	12.95%	12.76%	12.84%	13.41%	14.28%	17.06%	17.02%	17.00%	17.45%	16.55%
	0.2	11.21%	11.04%	11.44%	10.95%	11.95%	12.35%	13.77%	14.95%	15.00%	15.28%	14.78%
	0.3	10.36%	10.02%	9.94%	9.89%	10.13%	11.14%	12.30%	12.67%	12.69%	13.17%	12.22%
	0.4	9.35%	9.36%	**9.29%**	9.79%	10.20%	10.47%	11.74%	11.70%	11.67%	11.79%	10.92%
unweighted	0.0	13.19%	13.64%	11.27%	11.37%	11.54%	12.56%	13.36%	14.18%	14.17%	13.63%	13.85%
	0.1	8.81%	8.82%	8.65%	7.90%	8.81%	10.09%	11.72%	11.22%	11.22%	11.79%	11.64%
	0.2	8.28%	8.47%	8.33%	8.02%	8.41%	9.26%	10.23%	10.89%	10.93%	11.15%	10.63%
	0.3	7.93%	7.98%	8.07%	8.13%	8.32%	8.96%	9.76%	9.69%	9.67%	10.16%	10.31%
	0.4	8.10%	8.39%	8.54%	**7.82%**	8.50%	9.45%	9.78%	9.97%	9.94%	9.84%	10.37%

Table 4.4: Average gaps produced by the different versions of the *ADT* heuristic on the large instances of benchmarks H$^+$, `SW`, `Reg`, `Pla` and `Grid`. In dark grey the settings for which the p-value is ≥ 0.05 and in light grey those for which $0.05 > p \geq 10^{-4}$.

TADT (Table 4.5) applies the same rationale as *ADT* while limiting the computational time and, indeed, it obtains better gaps. For the weighed instances, the best average gap, that is 6.90%, is obtained by $\mu = 0.3$ and $\gamma = 0.2$, while the maximum one is 41.38%. For the unweighted instances, the best tuning is $\mu = 0.3$ and $\gamma = 0.1$ or 0.2, with an average gap equal to 5.06% and a maximum one of 52.94%. These settings are not only similar to each other, but also to those used by *SDT*. Moreover, as for *SDT*, the *RCL* mechanism dominates the *HBSS* one.

TADT		μ							α			
	γ	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0	1	2	3
weighted	0.0	10.62%	10.53%	9.76%	9.33%	9.79%	10.03%	11.02%	10.85%	10.85%	11.46%	11.20%
	0.1	8.54%	8.65%	7.74%	6.97%	7.87%	8.40%	9.13%	9.47%	9.49%	9.58%	9.29%
	0.2	8.58%	8.88%	7.54%	**6.90%**	7.25%	7.35%	7.98%	8.38%	8.33%	8.38%	8.39%
	0.3	8.59%	8.87%	7.77%	6.98%	7.22%	7.16%	7.31%	7.96%	7.92%	7.60%	7.74%
	0.4	8.82%	8.94%	7.82%	7.04%	7.22%	7.13%	7.34%	7.68%	7.68%	7.63%	7.83%
unweighted	0.0	7.97%	7.94%	6.38%	5.85%	5.95%	5.92%	6.60%	6.40%	6.42%	6.98%	6.58%
	0.1	7.17%	7.13%	5.99%	**5.06%**	5.49%	5.56%	6.07%	6.72%	6.74%	6.46%	6.59%
	0.2	6.99%	7.01%	6.08%	**5.06%**	5.32%	5.56%	5.97%	6.27%	6.25%	6.39%	6.48%
	0.3	6.99%	6.97%	6.06%	5.24%	5.40%	5.62%	6.00%	6.56%	6.60%	6.43%	6.32%
	0.4	7.00%	6.98%	6.03%	5.24%	5.45%	5.48%	5.75%	6.91%	6.88%	6.73%	6.48%

Table 4.5: Average gaps produced by the different versions of the *TADT* heuristic on the large instances of benchmarks `H`$^+$, `SW`, `Reg`, `Pla` and `Grid`. In dark grey the settings for which the p-value is ≥ 0.05 and in light grey those for which $0.05 > p \geq 10^{-4}$.

In summary, none of the considered tunings clearly dominates the other ones on all benchmarks, but the best performing algorithms (*SDT* and *TADT*) behave in a similar way, with the *RCL* variants performing better than the *HBSS* ones and the intermediate values of μ and γ better than the extreme ones. In particular, delaying the termination of the constructive phase is always profitable. One could also wonder whether the results obtained on the random graphs of benchmark `H`$^+$, with their peculiar structure (a single large safe component opposed to a single large unsafe one), could have an excessive impact on the parameter tuning. In practice, this is not the case: while the average gap on the other instances tends to be much larger, its dependence on the values of the parameters remains pretty much the same. In fact, most of the instances in benchmark `H`$^+$ contribute with a zero gap: they reduce the overall average, but do not influence the comparisons. Wilcoxon's test, in particular, automatically neglects such instances.

4.4.3 Tuning of the Scatter Search metaheuristic

The *SS* algorithm requires three parameters: the positive integer values n_B and n_D determine the size of the reference set, while the real coefficient $\mu \in [0,1]$ tunes the randomisation mechanism exploited by the *GRASP* algorithm that initialises it. For the sake of simplicity, we assume the number of best known and diverse reference solutions to be (at least nearly) the same: $n_B = \lceil R/2 \rceil$, $n_D = \lfloor R/2 \rfloor$, where R is the overall size of the reference set.

In order to determine the best tuning for parameters μ and R, we run the algorithm for 10 seconds on each of the large instances, that belong to the `SW`, `Reg`, `Pla` and `Grid` benchmarks (we did not consider `H`$^+$ because of the scarcely significant structure of the instances). The values of the parameters that we consider are $R \in \{5, 10, 15, 20, 30, 40\}$ and $\mu \in \{0.1, 0.2, 0.3, 0.4\}$. We estimate the quality of a heuristic solution S with the gap $(w(S) - w(\bar{S}))/w(\bar{S})$, where $\bar{S}$ is the best known solution for the given instance. When $w(\bar{S})$ is not optimal, this underestimates the actual gap. However, we shall see that for large instances neither the optimum nor a tight lower bound is known. Therefore, this is the most accurate estimate available and allows a comparison of different heuristic results.

Table 4.6 exhibits the results of the parameter tuning. There are two macro-columns: the first considers the weighted instances and the second the unweighted ones. Each row is associated with a value of R and each column with a value of μ. For each combination, we separately display the average gap over all weighted instances and all unweighted ones. We highlight the best values, that correspond to configuration $\mu = 0.2, R = 10$ for the weighted instances (labelled *SSw*) and $\mu = 0.1, R = 10$ for the unweighted ones (labelled *SSu*).

		weighted				unweighted			
		μ				μ			
		0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4
R	5	8.90%	7.25%	7.95%	8.14%	6.66%	6.64%	6.51%	7.75%
	10	8.54%	**6.76%**	7.43%	7.26%	**5.83%**	6.54%	6.48%	7.87%
	15	8.45%	7.23%	6.92%	8.02%	6.71%	6.44%	6.59%	8.17%
	20	8.43%	7.03%	8.01%	7.89%	6.58%	6.28%	7.09%	7.80%
	30	8.95%	7.64%	7.18%	8.46%	7.28%	6.78%	6.98%	8.76%
	40	8.73%	8.01%	8.70%	8.81%	7.18%	6.65%	6.89%	8.89%

Table 4.6: Average gaps produced by the Scatter Search heuristic after 10 seconds on all large instances for every configuration of the parameters.

Considering the maximum gap, instead of the average one, *SSw* is still the best version for the weighted instances, with a maximum gap of 26.74%. On the other hand, the configuration that minimises the maximum gap for unweighted instances is $\mu = 0.1, R = 20$, which achieved 20.51%, whereas *SSu* performs slightly worse with 23.08%. If we consider the whole benchmark, weighted and unweighted instances combined, *SSw* has the best average and maximum gaps: 6.65% and 28.21% respectively. Statistical tests confirm the existence of a group of configurations surrounding *SSw* and including *SSu*, whose performance does not show strongly significant differences. In Section 4.4.6, we will apply both *SSw* and *SSu* to all instances.

4.4.4 Comparison of the constructive-destructive approaches

After analysing the influence of the parameter tuning on the performance of all the considered algorithms, we proceed to compare the best versions of the alternative approaches, investigating the distribution of the gap values and distinguishing not only the weighted and unweighted instances, but also the single specific benchmarks.

In particular, for the weighted instances we compare *GRASP* with $\mu = 0.2$, *SDT* with $\mu = 0.3$ and $\gamma = 0.2$, *ADT* with $\mu = 0.2$ and $\gamma = 0.4$ and *TADT* with $\mu = 0.3$ and $\gamma = 0.2$, once again with a running time of 10 seconds on each instance. The average gaps reported in Table 4.3 suggest that *SDT* should perform better than *TADT*, whereas the other approaches should be worse than the previous ones, but more or less comparable with each other.

A more detailed comparison can be performed with the *SQD* diagram (Hoos and Stützle, 2004), that reports the fraction of instances for which the gap achieved does not exceed each possible value. Figure 4.1 provides the diagram of the four algorithms on the weighted large instances.

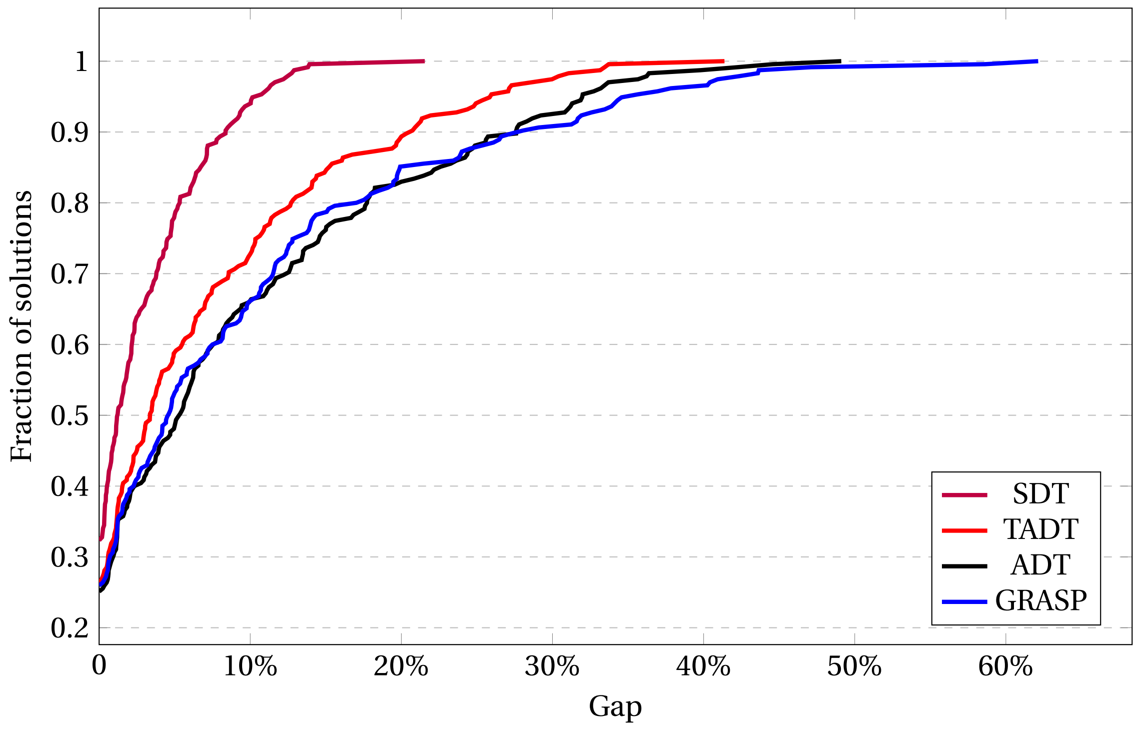

Figure 4.1: Solution Quality Distribution diagram over the large weighted instances of the presented heuristics (each using its best parameters). The diagram reports on the y-axis the fraction of instances such that each algorithm returns a solution whose gap (with respect to the best solution) is not worse than the value reported on the x-axis.

It shows a clear dominance of *SDT* over *TADT*, and of the latter over the other two algorithms. *GRASP* and *ADT* seem comparable, even if *GRASP* has worse results. Wilcoxon's test fully confirms these feelings: while the difference between *GRASP* and *ADT* is not statistically significant ($p \leq 0.083$ in favour of *GRASP*), all other relations have a p-value smaller than 10^{-11}. The number of best known results found is consistent with these results: 76 for *SDT*, 62 for *TADT*, 59 for *GRASP* and *ADT*.

As for the unweighted instances, we compare *GRASP* with $\mu = 0.4$, *SDT* with $\alpha = 1$ and $\gamma = 0.1$, *ADT* with $\mu = 0.3$ and $\gamma = 0.4$ and *TADT* with $\mu = 0.3$ and $\gamma = 0.2$. Figure 4.2 shows the corresponding *SQD* diagram. In this case, *GRASP* dominates *ADT* and both perform worse than *SDT* and *TADT*, which, however, do not clearly dominate each other. *SDT* seems more conservative, having less solutions with a large gap, but *TADT* finds a larger fraction of solutions with a small gap (less than 5%). According to Wilcoxon's test, *ADT* is significantly dominated by the other three algorithms ($p < 10^{-4}$) and *GRASP* is dominated by *TADT* ($p \leq 3.55 \cdot 10^{-5}$). The comparison of *SDT* with *GRASP* is weakly in favour of the former ($p = 0.06$), whereas we cannot say anything about *SDT* and *TADT*. As for the number of best know results found, this is slightly in favour of *TADT* (106) over *SDT* (96), while *GRASP* and *ADT* are worse, and nearly equivalent to each other (respectively, 87 and 86).

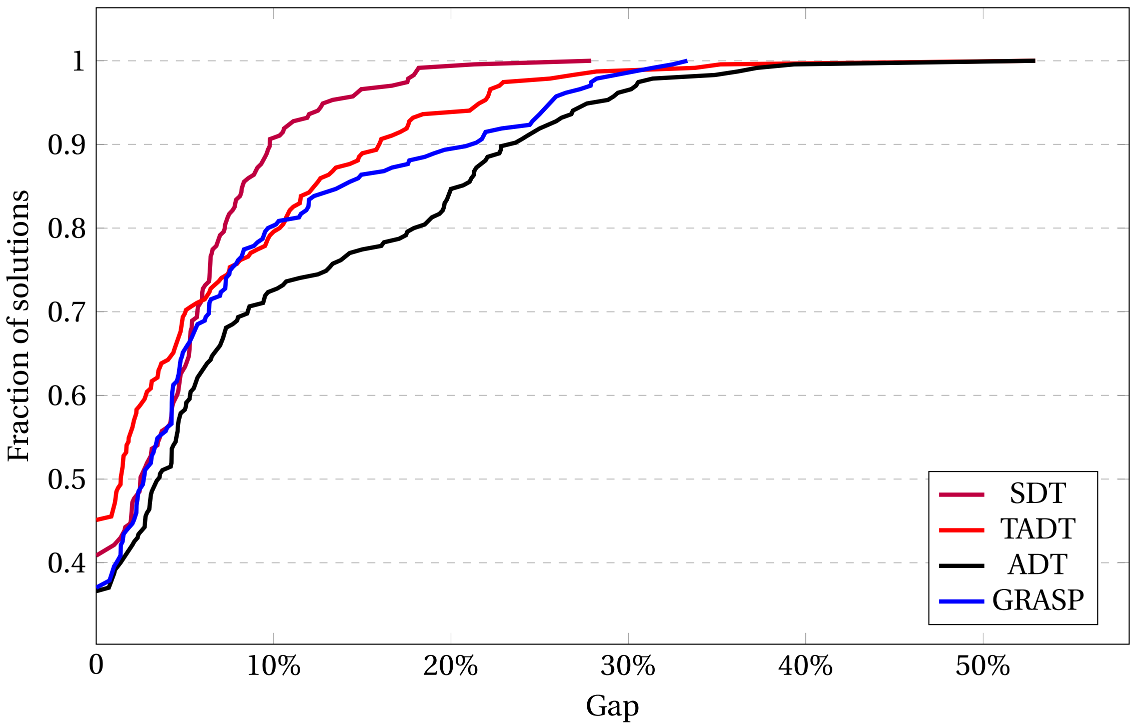

Figure 4.2: Solution Quality Distribution diagram over the large unweighted instances of the presented heuristics (each using its best parameters). The diagram reports on the y-axis the fraction of instances such that each algorithm returns a solution whose gap (with respect to the best solution) is not worse than the value reported on the x-axis.

The wide range of gaps observed reflects the very different performance that the algorithms exhibit on different classes of instances. This phenomenon is more pronounced in the *GRASP* metaheuristic than in the delayed termination algorithms, but clear in all cases. Table 4.7 describes in more detail the average gaps with respect to the best known results obtained by each algorithm on specific classes of instances. The left part of the table refers to the weighted instances, and the right part to the unweighted ones. In each part, four columns correspond to the four competing algorithms. A group of rows is associated with each benchmark, and further disaggregated with respect to a relevant structural parameter: benchmark H$^+$ is divided according to the density of the graph, benchmarks SW and Reg according to the (respectively, initial or exact) vertex degree, benchmark Grid is divided into 2D and 3D grids. We can see that *SDT* finds the best average gaps for most instance classes, but is less effective on the weighted grids and the unweighted regular graphs. Conversely, *TADT* has a good performance on these two classes. *ADT* and *GRASP* are less effective, apart from some classes of unweighted random graphs. This is consistent with the plots of Figures 4.1 and 4.2.

class		weighted				unweighted			
		GRASP	SDT	ADT	TADT	GRASP	SDT	ADT	TADT
H$^+$	0.1	2.63%	**0.67%**	3.00%	2.00%	1.08%	2.13%	1.41%	**0.30%**
	0.2	0.80%	**0.37%**	0.86%	0.65%	**0.16%**	0.41%	0.41%	0.24%
	0.3	0.26%	**0.25%**	0.34%	0.26%	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	0.4	0.05%	**0.00%**	0.05%	0.05%	**0.00%**	**0.00%**	**0.00%**	**0.00%**
	all	0.93%	**0.32%**	1.06%	0.74%	0.31%	0.63%	0.46%	**0.14%**
SW	6	30.46%	**5.56%**	15.92%	19.49%	14.80%	**7.97%**	18.97%	12.51%
	10	25.71%	**5.34%**	29.38%	20.51%	19.76%	**6.28%**	18.78%	12.55%
	all	28.09%	**5.45%**	22.65%	20.00%	17.28%	**7.13%**	18.88%	12.53%
Reg	5	10.98%	**3.53%**	12.73%	6.70%	2.92%	5.69%	4.49%	**2.63%**
	10	5.45%	**2.74%**	6.50%	4.55%	4.39%	5.38%	4.21%	**1.62%**
	all	8.21%	**3.13%**	9.62%	5.63%	3.66%	5.54%	4.35%	**2.12%**
Pla	all	10.12%	**6.29%**	14.47%	9.08%	6.34%	**7.39%**	18.15%	8.37%
Grid	2D	7.03%	5.22%	8.52%	**4.43%**	23.24%	**7.28%**	17.64%	31.27%
	3D	4.39%	4.64%	11.88%	**3.29%**	12.70%	**4.72%**	17.80%	15.38%
	all	5.71%	4.93%	10.20%	**3.86%**	17.97%	**6.00%**	17.72%	23.32%
all		9.44%	**2.84%**	9.29%	6.90%	6.03%	**4.01%**	7.82%	5.06%

Table 4.7: Average gaps produced by each heuristic (using its best parameters) on specific subsets of the large benchmark instances.

Since the real-world graphs of benchmark GC are rather heterogeneous, we analyse them individually. Table 4.8 is divided into an upper part dedicated to the weighted instances and a lower part dedicated to the unweighted ones. The first three columns provide the name of each instance, its number of vertices $|V|$ and of edges $|E|$. The remaining columns report, respectively, for the weighted and the unweighted instance, the value of the objective function for the solution returned by each of the four competing algorithms. The best result for each instance is bolded. The last row of each half of the table provides the average percent gap with respect to the best known result. While the smaller instances are solved with the same value by all algorithms, the larger ones show significant differences. The *TADT* heuristic hits the largest number of best known results (11 out of 18) and has the smallest average gap (0.92%), but in this benchmark *GRASP* comes second (with 10 best results and a gap equal to 2.17%) and the other two heuristics are worse, in particular in the unweighted case. This is in contrast with the better performance of *SDT*, observed on average for the other benchmarks, but consistent with the good performance of *TADT*.

	Instance	$\vert V\vert$	$\vert E\vert$	GRASP	SDT	ADT	TADT
weighted	adjnoun	112	425	155	**151**	156	153
	celegans_metabolic	453	2025	242	254	264	**239**
	celegansneural	297	2148	545	560	572	**528**
	dolphins	62	159	**83**	**83**	**83**	**83**
	football	115	613	243	**212**	225	215
	jazz	198	2742	**380**	418	431	388
	karate	34	78	**27**	**27**	**27**	**27**
	lesmis	77	254	**57**	**57**	**57**	**57**
	polbooks	105	441	**119**	127	**119**	127
	Avg.gap			2.42%	3.23%	4.63%	**1.29%**
unweighted	adjnoun	112	425	**31**	**31**	**31**	**31**
	celegans_metabolic	453	2025	49	53	**47**	48
	celegansneural	297	2148	90	93	97	**89**
	dolphins	62	159	**14**	15	**14**	**14**
	football	115	613	44	52	51	**40**
	jazz	198	2742	**85**	91	94	87
	karate	34	78	**6**	**6**	**6**	**6**
	lesmis	77	254	**11**	**11**	**11**	**11**
	polbooks	105	441	**24**	**24**	**24**	**24**
	Avg.gap			1.92%	7.68%	5.88%	**0.56%**

Table 4.8: Results obtained by the presented heuristics (with their best parameter tuning) in 10 seconds on the real-world graphs of benchmark GC

4.4.5 Comparison with the state of the art

In this section we apply the *SDT* algorithm, with its best tuning, to all instances available in the literature, that is to benchmarks `M` and `H`.

Considering the former, we compare *SDT* with the randomised destructive heuristic introduced in Macambira et al. (2019), that is denoted as *RD* in the following. The two algorithms run on different machines (the destructive heuristic is evaluated on an Intel Core TM i7-6700 with a 3.40 GHz CPU and 15.6 GB of RAM memory) and the computational times are quite short, which makes them easily subject to random fluctuations. We attempt a very approximate comparison through the conversion coefficient provided by PassMark Software (2022), according to which our machine should be 9224/8094 = 1.14 times faster than that employed by *RD*.

In order to try and perform a fair comparison, we have adopted the same approach of Macambira et al. (2019), fixing the number of restarts, instead of the computational time. Hence, both approaches run 1 000 iterations of their respective procedures. At each iteration, *RD* starts from the whole vertex set and removes one vertex at a time, generating a single minimal solution. Each iteration of *SDT* generates $\gamma|V|$ redundant solutions and reduces each of them to a minimal one.

Table 4.9 reports the results aggregated for instances with the same weight function and graph density δ, as specified in the first two columns. The third column provides the number of instances for each class. The following block of two columns (labelled "Domination") displays the number of instances for which each algorithm yields a result strictly better than its competitor. Then, the table provides the number of optima found by the two algorithms (all instances are solved exactly). The average and maximum gaps follow (with the better one in bold). The last block of two columns reports the average and the minimum value of the ratio between the computational times of *RD* and *SDT*.

			Domination		Optima		Average gap		Maximum gap		$\frac{t_{\text{RD}}}{t_{\text{SDT}}}$	
	δ	#	RD	SDT	RD	SDT	RD	SDT	RD	SDT	Avg.	Min.
weighted	0.3	21	0	**12**	9	**21**	5.37%	**0.00%**	20.24%	**0.00%**	3.22	1.74
	0.5	21	0	**15**	6	**15**	4.88%	**0.45%**	12.96%	**4.62%**	3.30	1.98
	0.7	21	0	**8**	10	**17**	1.80%	**0.58%**	11.71%	**5.29%**	3.41	2.34
	all	63	0	**35**	25	**53**	4.02%	**0.34%**	20.24%	**5.29%**	3.31	1.74
unweighted	0.3	21	0	0	21	21	**0.00%**	**0.00%**	**0.00%**	**0.00%**	4.48	1.74
	0.5	21	0	**2**	16	**18**	2.30%	**1.48%**	14.29%	14.29%	4.18	2.18
	0.7	21	0	**1**	17	**18**	1.73%	**1.20%**	11.11%	**9.09%**	4.09	2.28
	all	63	0	**3**	54	**57**	1.34%	**0.89%**	14.29%	14.29%	4.25	1.74
all		126	0	**38**	79	**110**	2.68%	**0.62%**	20.24%	**14.29%**	3.78	1.74

Table 4.9: Results obtained by 1000 iterations of *RD* and *SDT* on benchmark `M`. We display the number of instances for which one of the two algorithms performs better than the other, the number of optimal values found, the average and maximum gap with respect to the optimum and the (average and minimum) ratio of the computational times.

In the direct comparison, the *SDT* algorithm is always better (38 times) or at least as good (88 times) as *RD*. This is especially true for the weighted instances, where the strictly better results are 35 over 63 (in the unweighted ones, *RD* already found a large majority of the optimal results). Overall, *SDT* finds 110 optima versus 79, out of 126 instances. The average gap of *SDT* is better than that of *RD* over both the weighted and unweighted instances. As for the computational time, *SDT* is 3 to 4 times faster than *RD*. Even considering the 1.14 ratio suggested by PassMark Software (2022), the computational times are at least comparable.

Since benchmark `H` has been solved only with exact algorithms, and the performance of a heuristic cannot be meaningfully compared with them, we evaluate the ability of *SDT* to find in short time the optimal or best known results. In particular, the branch-and-bound algorithm B&Bs of Chapter 3 solves to optimality 254 out of the 280 instances, but requires one hour of computation on the same machine. Table 4.10 reports the results of *SDT* with a time limit of 60 seconds: each row corresponds to one of the possible sizes and each column to one of the four classes of density. The left part of the table concerns the weighted instances, and the right part the unweighted ones. Each cell contains the average gap with respect to the optimal, or the best known, solution.

	weighted				unweighted			
$\lvert V\rvert$	0.1	0.2	0.3	0.4	0.1	0.2	0.3	0.4
20	18.59%	-	-	-	-	-	-	-
25	3.70%	-	-	-	2.50%	-	-	-
30	-	-	-	-	-	-	-	-
35	-	-	0.24%	0.84%	-	-	-	2.50%
40	-	0.83%	-	2.31%	-	1.25%	4.44%	4.21%
50	1.37%	-	1.94%	3.05%	-24.77%	1.90%	2.61%	4.17%
60	-0.74%	-1.73%	2.93%	2.68%	-36.26%	-7.17%	0.69%	2.76%
avg.	3.28%	-0.13%	0.73%	1.27%	-8.36%	-0.57%	1.11%	1.95%

Table 4.10: Results obtained by *SDT* in 60 seconds on the small random instances of benchmark `H`.

Overall, the heuristic equals 193 results out of 280, concentrated in the smaller sizes (with the exception of the sparsest instances, whose optimal solutions seem hard to find). It worsens 68 results, mainly for denser instances of medium size. The gap tends to decrease for larger sizes, and finally becomes negative as 19 results on the sparser and larger instances are improved. The unweighted instances show remarkable gap reductions. While the heuristic is unable to find all the best results in short time, it gives however a useful contribution to reduce the optimality gap of the exact algorithm; in particular, 2 of the 26 open instances can now be solved running the branch-and-bound with the improved starting solution.

4.4.6 Comparison between the Scatter Search and the Delayed Termination heuristics

This section compares the Scatter Search heuristic with the *Simple Delayed Termination* (*SDT*) and the *Truncated Avoidant Delayed Termination* (*TADT*) heuristics (which are the best performing so far). Table 4.11 reports the average gaps produced by *SDT*, *TADT*, *SSu* and *SSw* for the various classes and sub-classes of large instances. The table is, once again, divided into two macro-columns separating the weighted instance from the unweighted ones. Each column is dedicated to a heuristic and each row to a sub-class of instances, with a last row providing the overall averages. We highlight in black the minimum gap among all heuristics for each sub-class.

Class	Sub	weighted				unweighted			
		SDT	TADT	SSu	SSw	SDT	TADT	SSu	SSw
	6	18.91%	34.60%	10.58%	**6.78%**	16.24%	18.40%	6.58%	**6.17%**
`SW`	10	**6.36%**	21.64%	15.23%	11.93%	11.46%	14.30%	**9.77%**	13.29%
	all	12.63%	28.12%	12.90%	**9.36%**	13.85%	16.35%	**8.17%**	9.73%
	5	**4.51%**	7.71%	6.79%	5.34%	3.49%	**2.73%**	2.99%	2.96%
`Reg`	10	**2.74%**	4.55%	4.59%	5.39%	4.46%	**1.62%**	3.90%	3.95%
	all	**3.62%**	6.13%	5.69%	5.37%	3.98%	**2.17%**	3.45%	3.46%
`Pla`	all	12.59%	15.49%	7.66%	**5.38%**	11.78%	10.89%	**6.05%**	6.14%
	2D	16.44%	15.60%	**3.07%**	5.19%	15.80%	38.43%	**8.94%**	11.83%
`Grid`	3D	4.68%	3.33%	3.37%	**3.22%**	20.85%	19.24%	**2.02%**	2.29%
	all	10.56%	9.47%	**3.22%**	4.20%	18.32%	28.83%	**5.48%**	7.06%
all		9.13%	16.26%	8.54%	**6.76%**	10.14%	11.01%	**5.83%**	6.54%

Table 4.11: Comparisons between the average gaps produced by each of the considered heuristics over different classes of large benchmark instances.

The results show that both versions of *SS* perform overall better than *SDT* and *TADT*, producing smaller gaps on the totality of the instances. The new heuristics are also more robust, as their maximum gaps are also smaller: 28.21% for *SSw* and 31.82% for *SSu* versus 36.08% for *SDT* and 62.50% for *TADT*. As well, they provide a larger number of best known solutions: 123 for *SSw* and 87 for *SSu* as opposed to 69 for *SDT* and 52 for *TADT*, out of 270. Indeed, *SSw* strictly improves 57 results and *SSu* improves 50 with respect to the previous best known ones. Wilcoxon's signed rank test (Wilcoxon, 1945) confirms the better performance of the two *SS* tunings with respect to *TADT* with p-values always smaller than 0.0005, but for *SDT* this only holds for the unweighted instances. For the weighted instances, in fact, the difference with respect to *SDT* is statistically insignificant for *SSu* and rather weak for *SSw* ($p \approx 6\%$). Delving into details, in fact, we can observe that for the regular instances (100 out of 270) the algorithms that produce the smallest gaps are *SDT* for the weighted case and *TADT* for the unweighted one. The special topology of this class, with its uniform value of the degree, undermines the ability of the *SS* metaheuristic, probably due to the role played by the degree in the *GRASP* initialisation procedure. Another sub-class of instances for which *SDT* performs better than *SS* is the small-world benchmark with initial degree 10, but only for unweighted instances.

For the sake of simplicity, in the following experiments we decided to select a single version of *SS*. All the indices above applied (average gap, maximum gap, number of best known solutions found and improved) concur in suggesting *SSw* as the better candidate.

4.4.7 Structure of the solutions and relation with the *Connected Safe Set Problem*

The literature on the *SSP* often discusses a variant, known as *Connected Safe Set Problem* (*CSSP*), in which the solution is required to consist of a single safe component (Fujita et al., 2014). We have already remarked in Section 4.4.1 that most of the best known solutions for random graphs consist of a single safe component opposed to a single unsafe one. We here extend the analysis of the structure of the best solutions to all large benchmarks.

Table 4.12 considers the same classes of instances addressed by Table 4.7. The first two columns specify the benchmark and class. The following two blocks of four columns refer, respectively, to the weighted and the unweighted instances. In each block, two columns, labelled $|\mathscr{C}_G(S)|$, provide the average and the overall range of values (minimum and maximum between square brackets) that the number of safe components assumes in the best known solution. The other two columns, labelled $|\mathscr{C}_G(V \setminus S)|$, provide the same information for the unsafe components. We refer to the best known solutions because they are the only possible approximation of the unknown optimal ones.

		weighted				unweighted			
		$\lvert\mathscr{C}_G(S)\rvert$		$\lvert\mathscr{C}_G(V \setminus S)\rvert$		$\lvert\mathscr{C}_G(S)\rvert$		$\lvert\mathscr{C}_G(V \setminus S)\rvert$	
class		Avg.	Range	Avg.	Range	Avg.	Range	Avg.	Range
	0.1	1	[1,1]	3	[2,6]	1	[1,1]	2.2	[1,5]
	0.2	1	[1,1]	1.52	[1,3]	1	[1,1]	1.32	[1,3]
H$^+$	0.3	1	[1,1]	1.2	[1,2]	1	[1,1]	1	[1,1]
	0.4	1	[1,1]	1.04	[1,2]	1	[1,1]	1	[1,1]
	all	1	[1,1]	1.69	[1,6]	1	[1,1]	1.38	[1,5]
	6	1	[1,1]	5.36	[4,7]	1.12	[1,2]	4.44	[3,8]
SW	10	1	[1,1]	3.68	[3,5]	1	[1,1]	3.08	[3,4]
	all	1	[1,1]	4.52	[3,7]	1.06	[1,2]	3.76	[3,8]
	5	1	[1,1]	7.04	[3,16]	1.48	[1,3]	9.76	[2,19]
Reg	10	1.08	[1,2]	7.08	[2,18]	1.36	[1,2]	7.56	[2,18]
	all	1.04	[1,2]	7.06	[2,18]	1.42	[1,3]	8.66	[2,19]
Pla	all	1	[1,1]	7.92	[5,13]	1	[1,1]	6.36	[4,10]
	2D	1	[1,1]	4.8	[4,7]	1	[1,1]	3.4	[3,4]
Grid	3D	1	[1,1]	4.8	[3,8]	1	[1,1]	3	[2,6]
	all	1	[1,1]	4.8	[3,8]	1	[1,1]	3.2	[2,6]
all		1.01	[1,2]	4.23	[1,18]	1.10	[1,3]	4.04	[1,19]

Table 4.12: Structure of the best known solutions for all classes of instances.

The clearest observation is that a large majority of solutions have a single safe component (445 out of 470), while 24 have 2 components and only one has 3. In short, the *CSSP* has very frequently the same solution as the basic *SSP*. Considering the unsafe components, the random instances exhibit the behaviour already described in Section 4.4.1: the best known solution is often complemented by a single unsafe component, and this becomes more and more frequent as the size and density of the instances increases. The sparse instances show a wider range of values, but still the number of unsafe components is often equal to 1 (in 140 instances out of 470) and most of the time less than 10 (in more than 400 instances). The larger values seem to occur in the regular instances, but it is presently impossible to determine whether this actually reflects the structure of the optimal solutions for such instances, or the fact that the best known results are actually far from optimal. The same results are illustrated in Figure 4.3 which displays the number of safe components (in blue) and unsafe components (in red) for each of the 235 large instances. In order to improve readability, the instances have been sorted in ascending order of the number of safe components (in the first case) or of the number of unsafe components (in the second case) in the best known solution. The figure can also be understood as the cumulative number of instances for which the best known solution has a number of safe (or unsafe) components lower than a threshold given on the vertical axis.

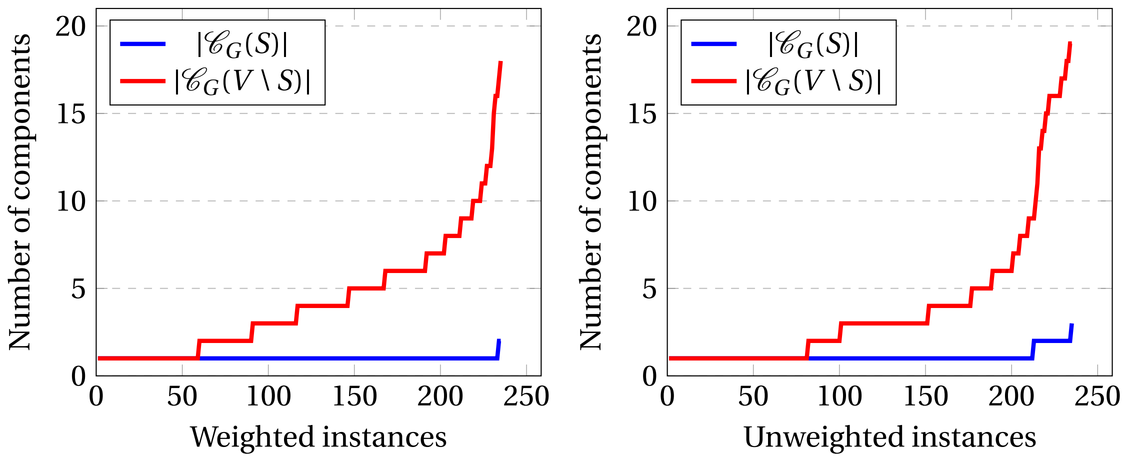

Figure 4.3: Number of safe and unsafe components in each instance, for both weighted (left) and unweighted instances (right).

4.4.8 Heuristic initialisation of the branch and bound

We now examine the impact of heuristic initialization on the performance of the branch-and-bound algorithm presented in Chapter 3.

Since it was the heuristic that performed best, we decided to adopt *SS* to generate an initial incumbent solution for the B&Bv (see Section 3.5) algorithm. Tables 4.13 and 4.14 share the same structure of Tables 3.9 and 3.10, but compare the results obtained without the heuristic initialisation (B&Bv) or with the most robust and versatile parameter setting of SS (SSw+B&Bv). Since the smaller instances are all solved to optimality within seconds (the longest computation for $|V| \leq 35$ lasts 20.079 seconds) we decided to focus only on the instances with $|V| \geq 40$, which pose a significant challenge to the algorithm. The time limit is still of one hour, including the time spent on the heuristic. The heuristic is run for 0.1 seconds for $|V| = 40$, 1 second for $|V| = 50$ and 10 seconds for $|V| = 60$. The rationale for this choice is that it makes sense to increase the time limit as the size of the instance grows.

The gaps reported in the following tables are not computed with (20) as in the previous sections but with (19).

		B&Bv				SSw+B&Bv			
δ	$\lvert V\rvert$	gap	solved	CPU	BN	gap	solved	CPU	BN
	40	-	5	2.788	299276	-	5	2.391	244278
0.1	50	-	5	111.651	8102922	-	5	99.951	7217037
	60	-	5	1199.706	62877972	-	5	844.651	43528549
	40	-	5	3.997	277123	-	5	3.763	251329
0.2	50	-	5	307.728	14711729	-	5	285.167	13837516
	60	1.82%	3	2290.738	74773767	1.07%	3	2217.319	74164033
	40	-	5	4.898	266300	-	5	4.940	265918
0.3	50	-	5	47.728	1665147	-	5	44.492	1554477
	60	-	5	743.674	17805962	-	5	648.941	15708243
	40	-	5	2.059	89991	-	5	2.302	96953
0.4	50	-	5	27.720	773785	-	5	26.165	713841
	60	-	5	271.408	5005143	-	5	270.733	4837895

Table 4.13: Comparison between the B&Bv algorithm without and with an initial heuristic solution over all the weighted instances of the benchmark.

Considering the weighted instances (see Table 4.13), the number of unsolved ones remains 2 out of 60, but for one of them both the lower and the upper bound improve. The computational time and the number of branching nodes tend to decrease, but not uniformly. The denser instances with $|V| = 40$ and $|V| = 60$, in fact, show a moderate increase. In the unweighted case (see Table 4.14), the number of unsolved instances decreases from 16 to 15 and the number of memory overflows from 10 to 3. The average gap sharply decreases on the sparse instances, as both the lower and the upper bound improve. For $\delta \in \{0.1, 0.2\}$, the computational times and the branching nodes are significantly lower. For $\delta = 0.3$, however, the decrease is weak and for $\delta = 0.4$ there is a moderate increase, as in the weighted case.

		B&Bv				SSw+B&Bv			
δ	$\lvert V\rvert$	gap	solved	CPU	BN	gap	solved	CPU	BN
0.1	40	-	5	44.817	4778665	-	5	25.667	2718502
	50	1.25%	4	2165.374	152910911	-	5	1242.201	92944220
	60	68.40%	0	OM	OM	14.40%	0	3600.000*	189966296*
0.2	40	-	5	47.749	3390965	-	5	23.600	1821474
	50	-	5	1974.602	100802247	-	5	1320.257	72594184
	60	33.90%	0	OM	OM	10.40%	0	3600.000	131862708
0.3	40	-	5	14.626	837818	-	5	13.135	782608
	50	-	5	480.459	18963773	-	5	435.720	18325922
	60	4.29%	0	3600	94802520	4.29%	0	3600.000	98738374
0.4	40	-	5	5.8	270571	-	5	5.515	266566
	50	-	5	56.295	1689201	-	5	60.285	1862514
	60	-	5	710.371	13912611	-	5	789.761	15958696

Table 4.14: Comparison between the B&Bv algorithm without and with an initial heuristic solution over all the unweighted instances of the benchmark.

In summary, a good heuristic initialisation gives a useful contribution mainly on the sparser instances, that are the harder ones for the exact algorithm.

To test the limits of the algorithm, we feed it the instances with $|V| = 100$ from the `SW`, `Reg` and `Pla` benchmarks, that had been previously tackled only by heuristic approaches. Given the larger size, we have set the time limit for the heuristic initialisation to 60 seconds. We excluded the `Grid` benchmark because it includes only two instances with 100 vertices. Unfortunately, this size proved challenging for the exact algorithm: all instances caused a memory overflow before hitting the time limit of one hour.

Table 4.15 reports the results. Its first two columns identify the main classes and the sub-classes of instances, the third one reports the average gap on the weighted instances (as usual, 5 instances for each row) and the last column does the same for the unweighted instances.

Class	Sub-class	weighted	unweighted
`SW`	6	67.97%	116.21%
	10	70.36%	117.44%
`Reg`	5	122.00%	370.61%
	10	105.38%	193.97%
`Pla`		40.93%	75.90%

Table 4.15: Average gaps produced by the SS+B&Bv algorithm over the instances with $|V| = 100$ after reaching the memory limit.

The weighted instances are confirmed to be easier than the unweighted ones. The most difficult instances are the regular ones, followed by the small-world instances and, finally, the planar ones. These results suggest that further improvements would probably require the development of a tighter lower bound, to avoid generating a huge number of branching nodes.

Chapter 5

Hard Interdiction Problems

In this chapter, we describe the class of binary interdiction problems that we call hard interdiction. In these problems there are two agents which are called attacker *and* defender *which aim to optimise the same objective in opposite directions. The attacker disrupts the defender's instance by blocking elements so that the latter's optimal solution has the worst possible objective value. We introduce the notation for these problems and discuss the main results from the literature. Then we focus our attention on two specific hard interdiction problems and their respective properties.*

Binary Interdiction Problems are bilevel combinatorial optimisation problems that can be described as zero-sum Stackelberg games with two agents: the leader (also called *attacker*) and the follower (also called *defender*). The follower's goal is to solve a combinatorial optimisation problem for a given instance where the attacker might have interdicted some elements, that is, simply blocked them or made them more expensive. We denote the former case as *Hard Interdiction Problems (HIP)* and the latter as *Soft Interdiction Problems (SIP)*. On the other hand, the attacker, who is the first to move, has to decide which elements to interdict, considering that each element incurs an interdiction cost and the total cost of the chosen elements must not exceed a given budget. In this thesis we focus on HIP.

5.1 Notation

Without loss of generality, we consider a generic interdiction problem in the max-min form, where the attacker aims to maximise the defender's feasible solution of minimum cost. Throughout the thesis we will adopt the following notation:

- Y is the set of all feasible solutions for the defender (also called *responses*);
- $c : Y \to \mathbb{R}^+$ is the objective value of the responses;
- I is the set of all interdictable elements;
- $g : I \to \mathbb{R}^+$ is the interdiction cost of each element in I;
- $B \in \mathbb{R}^+$ is the interdiction budget;
- $X_B := \left\{x \in \wp(I) | \sum_{i\in x} g_i \le B\right\}$ is the set of all feasible solutions for the attacker (also called *attacks*), where $\wp$ is the power set function;

To simplify the discussion, we also introduce some possible abuses of notation:

- given an interdictable item $i \in I$ and a feasible defender's solution $y \in Y$, we indicate by $i \in y$ the fact that, if i is interdicted, then y is no longer feasible; on the other hand, $i \notin y$ means that the interdiction of i does not affect the feasibility of y;
- consequently, given feasible solutions $x \in X_B$ and $y \in Y$ for the two agents, $x \cap y = \left\{i \in x | i \in y\right\}$ is the set of elements of the former that makes the latter infeasible;
- $\forall Y' \subseteq Y : \forall x \in X_B : Y'(x) := \left\{y \in Y' | x \cap y = \emptyset\right\}$ is the set of solutions in Y' that remain feasible after attack x; in particular, $Y(x)$ is the set of *all* defender's solutions that are feasible after attack x;
- $\forall x \in X_B : \ c(x) := \min_{y\in Y(x)} c(y)$ denotes the objective value of the best defender's solution after attack x.

Now we can formulate the HIP as follows

$$HIP: \quad \max_{x\in X_B} c(x) = \max_{x\in X_B} \min_{y\in Y(x)} c(y) \tag{21}$$

Throughout the thesis we will address $x^* := \arg\max_{x\in X_B} c(x)$ as the optimal solution of HIP and $c(x^*)$ as its value. We will also assume the availability of an exact procedure to solve the defender's problem for any given attack x. In this context, any feasible attack found during the computation is denoted as an *incumbent* solution and provides a lower bound on $c(x^*)$, just like in a classical branch-and-bound approach. On the other hand, computing an upper bound on $c(x^*)$ is more complicated.

5.2 Literature review

The first interdiction problem considered in literature is due to Wollmer (1964) that considered the Maximum Flow Interdiction Problem (MFIP): given a directed graph $G = (N, A)$ with a source node $s \in N$ and a sink node $t \in N$, arc capacities $c : A \to \mathbb{R}^+$, interdiction costs $g : A \to \mathbb{R}$ and a positive attack budget $B \in \mathbb{R}^+$, compute a feasible attack $x \subseteq A$, subject to $\sum_{(i,j)\in x} g_{ij} \leq B$, such that the maximum $s - t$ flow in graph $G_x = (N, A \setminus x)$ with capacities c is minimised. For this problem, the author proposes a simple exact algorithm. A military application of the MFIP is considered by Ghare et al. (1971) that also provide a branch-and-bound algorithm for the problem. Only some decades after, Phillips (1993) proves the NP-hardness of the MFIP.

Other problems arise after the MFIP, like a non-binary interdiction version of the shortest path problem by Fulkerson and Harding (1977) that they prove to be equivalent to a parametric version of the minimum cost flow problem. Then the shortest path interdiction problem (hard version) is presented in Malik et al. (1989) as well as a flawed algorithm for its computation. In fact, as pointed out in Section 5.2 of Israeli and Wood (2002), the authors made assumptions that turned out to be incorrect. Nonetheless, Israeli and Wood (2002) themselves correct these mistakes by proposing two solution methods that are still used today to solve interdiction problems. The complexity of this problem is proved to be NP-hard by Ball et al. (1989).

Moreover, other interdiction problems are considered in the literature like the Maximum Clique Interdiction Problem (Rutenburg, 1991; Furini et al., 2019), the Bilevel Knapsack Interdiction Problem (Caprara et al., 2013, 2014, 2016; Fischetti et al., 2019; Della Croce and Scatamacchia, 2020; Weninger and Fukasawa, 2023), various facility location interdiction problems (Church et al., 2004; Scaparra and Church, 2008; Fischetti et al., 2019; Fröhlich and Ruzika, 2021; Lunday, 2024), the minimum cut interdiction problem (Abdolahzadeh et al., 2020), the electric power grid interdiction problem (Church et al., 2004), and many other problems.

Real world applications Given the adversarial nature of these problems, they are capable of accurately modelling a wide range of real-world situations. For example, Ghare et al. (1971) propose a military application to interdict some supply routes of an invader by considering the MFIP. Similarly, Morton et al. (2007) propose an intel application for disrupting the supply network of illegal items such as nuclear weapons. Changing typology, Assimakopoulos (1987) proposes an interdiction model for the containment of infections inside hospitals. Another possible application is presented by Mansi et al. (2012) that model a financial investments planning as an interdiction problem. Numerous other practical applications can be modelled using interdiction problems.

Computational complexity The complexity of these problems varies significantly depending on the specific problem taken into account. In fact, according to the status of the follower's problem, two main cases emerge. When the follower's problem is polynomial, the interdiction problem is in NP and often NP-complete. Examples of such NP-complete problems are the Shortest Path Interdiction Problem and the Maximum Flow Interdiction Problem. When the follower's problem is NP-complete, the interdiction often ends up being Σ_2^P-complete, as happens for example for the Bilevel Interdiction Knapsack Problem and the Maximum Clique Interdiction Problem (Caprara et al., 2013; Rutenburg, 1991) and several other interdiction problems considered in Grüne and Wulf (2023).

Solution methods Some authors presented ad-hoc formulations for specific problems. See for example Church et al. (2004) and Scaparra and Church (2008) for facility location. Other approaches can be applied to more general cases.

When the defender's problem is polynomial, it is possible to exploit the *dualise-and-combine* technique. The dualise-and-combine is a technique the consists in manipulating the dual formulation of the defender's problem and directly applying the binary interdiction variables since now both levels share the same optimisation directions. This is only possible (assuming that $P \neq NP$) when the defender's problem is polynomial because we have to be able to model the problem with continuous variables to dualise the formulation.

However, many interdiction problems do not satisfy such requirements and sometimes even if they do, the dualise-and-combine method does not yield efficient approaches. Therefore, in the literature the most common techniques to solve these problems are *Benders decomposition* and *super-valid* (covering) inequalities (Wood, 1993; Israeli and Wood, 2002). The former is designed for SIP but can also be applied to HIP, and consists in reformulating the problem as a single level formulation with a combinatorial number of constraints that are associated to every defender's feasible solutions. This way, it is possible to express the interdiction as an increment of the objective value of the solutions that are affected by the attack. The super-valid inequalities are covering constraints that impose that some good quality defender's feasible solution has to be interdicted. They can be applied to a single level reformulation in order to strengthen them, or can be used independently, in which case we refer to them as *covering decomposition*. All these methods will be better explained in Section 6.1. It is worth mentioning that Fischetti et al. (2019) define a class of interdiction problems that enjoy the so called *downward monotonicity* property which allows to reduce HIP into an equivalent SIP version without resorting to large coefficients.

For more details about interdiction problems, models and algorithms we refer the interested reader to Smith and Song (2020).

5.2.1 Fortification problems

Recently, the focus of research in this area has shifted to *Binary Fortification Problems* (BFP). These problems are a natural extension of BIPs that introduce an additional initial turn in which the defender spends a *protection budget* $P \in \mathbb{R}^+$ to immunise some elements. In details, for every interdictable element $i \in I$ there is a *fortification cost* $p_i \in \mathbb{R}^+$ and the fortification problem consists in computing a subset of interdictable elements $w \subseteq I$ such that $\sum_{i \in w} p_i \leq P$ and these elements cannot be interdicted.

In other words, if $W = \{w \subseteq I | \sum_{i \in w} p_i \leq P\}$ is the set of feasible *fortifications* and $X_B(w) := \{x \in X_B | x \cap w = \emptyset\}$ is the set of feasible attacker's solutions which does not interdict the elements fortified by w, the generic BFP can be formulated as

$$BFP: \quad w^* = \arg\min_{w \in W} c(w) = \arg\min_{w \in W} \max_{x \in X_B(w)} \min_{y \in Y(x)} c(y) \tag{22}$$

and w^* is the optimal solution.

To the best of our knowledge, the first fortification problem presented in the literature is due to Brown et al. (2006) where the authors describe a situation that involves deciding what components to fortify in an electric grid targeted by some terrorists. Moreover, problems like the shortest path fortification problem are tackled by Cappanera and Scaparra (2011); Lozano and Smith (2017); Leitner et al. (2023) where the defender can immunise some arcs that the attacker won't be able to interdict in order to contain the damage.

The practical applications of these problems can be easily found in the effort to find the best way to protect a system (typically a network) from some malicious attacker.

The complexity of such problems is even higher than that of interdiction counterpart. In fact, Boggio Tomasaz et al. (2024) show that many fortification problems are Σ_2^P-complete even if the last-level problem is polynomial. Moreover, the complexity of generalized multi-level interdiction problems seems to confirm the conjecture that as the number of levels increases, the complexity of the problem rises to the next higher class in the polynomial hierarchy. This is confirmed in the specific case of the multi-level knapsack interdiction problem (even with unit costs), for which Boggio Tomasaz et al. (2024) demonstrate the Σ_k^P-completeness for every possible number of levels k.

It is easy to notice that fortification problems are a form of HIP applied to another interdiction problem. Therefore, any algorithmic framework for HIP also works for fortification problems. In fact, the main exact approaches for these problems consist of *single level reformulations* based on Benders decomposition and *super valid inequalities.* Notice that for these multi-level problems with 3 or more levels, it is not possible to exploit the dualise-and-combine technique since their complexity is higher than NP. On the other hand, since the attacker's problem has a knapsack structure, it is possible to exploit the *downward monotonicity* property to efficiently convert the higher level problem from HIP to SIP. This last expedient is exploited in Leitner et al. (2023) that propose a branch-and-cut algorithm for any fortification problem.

5.3 Two representative problems

In this thesis we focus on two HIPs: the hard version of the *Shortest Path Interdiction Problem* (SPIP) also commonly referred to as "k-most-vital arc problem"; and the *Set Covering Interdiction Problem* (SCIP).

These problems share certain traits but also exhibit key differences: The SPIP has been extensively studied in the literature and represents a classic case of interdiction where the defender's problem is polynomial; on the other hand, the SCIP is a new problem (although similar problems can be found in the literature) where the defender's problem is NP-complete.

For both problems, we provide a formal description along with any relevant properties. In particular, we demonstrate how to recognize degenerate instances where the attack budget becomes sufficient to completely interdict the defender's solution space, Y. As the attack budget B increases, the attacker's solution space, X_B, expands monotonically, leading to progressively better outcomes for the attacker (and worse for the defender). This progression begins at $B = 0$, where the only feasible attack is the trivial empty set $\emptyset$, reducing the problem to that of the defender. It culminates when B reaches a point where an attack becomes feasible that interdicts all of the defender's solutions. Beyond this point, further increasing the budget yields no additional benefit for the attacker, as the optimal outcome has already been achieved.

5.3.1 Shortest Path Interdiction Problem

Let $G = (N, A)$ be a directed graph (or the directed symmetric version of an original undirected graph) with a cost function $c : A \to \mathbb{R}^+$ over the arcs, two special nodes $s, t \in N$, interdiction costs $g : A \to \mathbb{R}^+$ and an attack budget $B \in \mathbb{R}^+$. The SPIP looks for a subset of arcs $x \subseteq A$ subject to $\sum_{(i,j)\in x} g_{ij} \leq B$ such that the shortest path from s to t in the graph $G_x = (N, A \setminus x)$ is maximised.

Similarly to the MFIP, the SPIP has military defence applications in the disruption of supply routes of a potential invader.

The computational complexity has already been discussed in Section 5.2, where it was shown that, since the shortest path problem is polynomial, the SPIP is NP-complete (Ball et al., 1989). Moreover, since the defender's problem is polynomial, it is possible to apply the dualise-and-combine technique to obtain the following well-known formulation for the SPIP.

$$
\begin{aligned}
\max \quad & d_t && && (23.1)\\
\text{s.t.} \quad & && && \\
& d_j - d_i \leq c_{ij} + M \cdot x_{ij} && (i,j) \in A && (23.2)\\
& d_s = 0 && && (23.3)\\
& \sum_{(i,j)\in A} g_{ij} \cdot x_{ij} \leq B && && (23.4)\\
& d_i \geq 0 && i \in N && (23.5)\\
& x_{ij} \in \{0,1\} && (i,j) \in A && (23.6)
\end{aligned}
$$

where the intuitive meaning of the variables is:

- $x_{ij} = 1$ if and only if arc (i, j) is interdicted;
- d_i is the distance (in terms of c) from s to i in the interdicted instance.

As we can see, the formulation is almost identical to the classical dual formulation for the shortest path problem. The presence of the binary variables x only affects the model in the budget constraint (23.4) and in the classical constraint (23.2) where, instead of the usual cost c_{ij}, the right-hand-side is $c_{ij} + M \cdot x_{ij}$. The idea is that by interdicting the arc (i, j), its cost increases by some very large constant M that discourages the defender from choosing a path including such arc. In practice M has to be larger than the optimal solution of the SPIP.

As anticipated, to check whether an instance is trivial (can be completely interdicted) we can exploit a very simple idea:

Remark 3. *A minimum $s-t$ cut (using the interdiction costs g as capacities) is the minimum interdiction cost attack that is able to interdict all $s-t$ paths in G.*

Therefore, we can compute the minimum cut in polynomial time and, if its value is smaller than or equal to the attack budget B, the minimum cut itself is the optimal attack for the SPIP.

In the literature, Malik et al. (1989) tried to exploit this relationship between the SPIP and the minimum cut problem even further by computing a minimum cut of a sub-graph of G to interdict a subset of $s-t$ paths. However, the flaw was that they assumed that the minimum attack that interdicts a subset of paths is equal to the minimum cut of the sub-graph induced by the considered paths. This was proved wrong by Israeli and Wood (2002) that provide a counterexample. Another interesting aspect about the relationship between the SPIP and flow-related problems is that the real-valued version of the SPIP (the increment in the cost is not binary but continuous) can be reduced to a computation of a minimum cost flow problem (Fulkerson and Harding, 1977).

Lagrangean relaxation and min-cost flow An interesting fact that we discovered, is that the Lagrangean relaxation of the budget constraint in formulation (23) for the SPIP is strictly related to the min-cost flow problem.

The relaxation assumes the form

$$\max \quad d_t - \lambda \cdot \left(\sum_{(i,j)\in A} g_{ij} \cdot x_{ij} - B \right) \equiv d_t - \lambda \cdot \sum_{(i,j)\in A} g_{ij} \cdot x_{ij} \tag{24.1}$$

s.t.

$$d_j - d_i \le c_{ij} + M \cdot x_{ij} \qquad (i,j) \in A \tag{24.2}$$

$$d_s = 0 \tag{24.3}$$

$$d_i \ge 0 \qquad i \in N \tag{24.4}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A \tag{24.5}$$

where we discarded the $+\lambda \cdot B$ term in the objective because it is a constant factor.

Now, let's consider the classical formulation for the problem of computing a minimum cost flow of value $W > 0$ from s to t with capacities g. The costs of the arcs are c and the value W, for now, is arbitrary. We denote with $\delta_k^- := \{i \in V \mid (i,k) \in A\}$ the in-neighbourhood and with $\delta_k^+ := \{j \in V \mid (k,j) \in A\}$ the out-neighbourhood.

$$\begin{aligned} \min \quad & \sum_{(i,j)\in A} c_{ij} \cdot y_{ij} \\ \text{s.t.} \quad & \\ & \sum_{i\in\delta_k^-} y_{ik} - \sum_{j\in\delta_k^+} y_{kj} = \begin{cases} W & \text{, if } k = t \\ -W & \text{, if } k = s \\ 0 & \text{, otherwise} \end{cases} & k \in V \\ & 0 \le y_{ij} \le g_{ij} & (i,j) \in A \end{aligned}$$

and its dual formulation

$$\max \quad W \cdot d_t - \sum_{(i,j)\in A} g_{ij} \cdot r_{ij} \tag{25.1}$$

s.t.

$$d_j - d_i \le c_{ij} + r_{ij} \qquad (i,j) \in A \tag{25.2}$$

$$d_s = 0 \tag{25.3}$$

$$d_i \ge 0 \qquad i \in N \tag{25.4}$$

$$r_{ij} \ge 0 \qquad (i,j) \in A \tag{25.5}$$

The structure is very similar to that of formulation (24) except for the variables r_{ij} instead of $M \cdot x_{ij}$ and the coefficients in the objective: W multiplies d_t and there is no λ that multiplies the sum. If we consider the continuous relaxation of formulation (24), we can see that if M is an upper bound of the variables r_{ij}, we can perform a variable change $r_{ij} = M \cdot x_{ij}$. This way, constraints (24.2) and (25.2) become identical. Moreover, now the objective becomes

$$\max \quad d_t - \lambda \sum_{(i,j)\in A} g_{ij} \cdot \frac{r_{ij}}{M} = d_t - \frac{\lambda}{M} \sum_{(i,j)\in A} g_{ij} \cdot r_{ij}$$

and, to adjust the coefficient of d_t, we multiply the objective by $W > 0$, obtaining

$$\max \quad W \cdot d_t - \frac{W \cdot \lambda}{M} \sum_{(i,j)\in A} g_{ij} \cdot r_{ij}$$

but since W is an arbitrary parameter, we can set $W = \frac{M}{\lambda}$ to obtain

$$\max \quad W \cdot d_t - \sum_{(i,j)\in A} g_{ij} \cdot r_{ij} = \frac{M}{\lambda} \cdot d_t - \sum_{(i,j)\in A} g_{ij} \cdot r_{ij}$$

which is exactly the objective (25.1).

This result is consistent with Fulkerson and Harding (1977) where they show that the continuous version of the soft shortest path interdiction is equivalent to the minimum cost flow problem.

5.3.2 Set Covering Interdiction Problem

In order to provide a simple description, we use the *binary matrix* version of the problem. Given a binary matrix $A \in \{0,1\}^{m \times n}$ with associated sets of column indices $\mathcal{I} = \{1,\dots,n\}$ and row indices $\mathcal{J} = \{1,\dots,m\}$, and a cost function $c : \mathcal{I} \to \mathbb{R}^+$, a feasible defender's solution y is a subset of columns of A such that the sub-matrix induced by the columns of y contains at least a 1 in every row: $Y = \{y \subseteq \mathcal{I} | \forall j \in \mathcal{J} : \exists i \in y : A_{ji} = 1\}$ is the set of all feasible solutions. The defender aims to find the feasible solution y^* which minimises the sum of the column costs. On the other hand, given an interdiction cost $g : \mathcal{I} \to \mathbb{R}^+$ on each column and a budget $B \in \mathbb{R}^+$, the attacker aims to block a subset of columns $x \subseteq \mathcal{I}$ with $\sum_{i \in x} g_i \leq B$, so that the optimal response of the defender, that does not use the interdicted columns, has maximum cost.

The SCIP, unlike the SPIP, does not have a polynomial defender's problem, therefore it is not possible to apply the dualise-and-combine unless $P = NP$. Moreover, we believe it to be Σ_2^P-complete, therefore we cannot provide a compact ILP formulation unless $P = NP$.

As for the SPIP, we now discuss the possibility to forbid all defender's solutions. Let

$$\begin{aligned} C_j &:= \{i \in \mathcal{I} | A_{ji} = 1\} && \forall j \in \mathcal{J} \\ R_i &:= \{j \in \mathcal{J} | A_{ji} = 1\} && \forall i \in \mathcal{I} \end{aligned}$$

be the set of columns that cover a given row $j \in \mathcal{J}$ and the set of rows that are covered by a given column $i \in \mathcal{I}$.

Proposition 9. *The SCIP admits an attack x^* that forbids all defender's solutions if and only if there is some row $j \in \mathcal{J}$ such that the sum of the interdiction costs of the columns C_j respects the budget.*

$$\exists x^* \in X_B : Y(x^*) = \emptyset \iff \exists \jmath \in \mathcal{J} : \sum_{i \in C_\jmath} g_i \leq B$$

Proof. Suppose we know some feasible attack $x^* \in X_B$ that can interdict the entire Y. Let's consider the defender's solution $\hat{y} = \mathcal{I} \setminus x^*$ of all columns except the ones interdicted. Since x^* interdicts all feasible solutions, $\hat{y}$ must be an infeasible set of columns. Therefore, there must be some row $\jmath \in \mathcal{J}$ which is not covered by $\hat{y}$. Since $\hat{y}$ contains every column except the interdicted ones, it means that all columns $C_\jmath$ have been interdicted, so $C_\jmath \subseteq x^*$ and $\sum_{i \in C_\jmath} g_i \leq \sum_{i \in x^*} g_i$. Finally, since $x^* \in X_B$, we know that $\sum_{i \in x^*} g_i \leq B$ which implies $\sum_{i \in C_\jmath} g_i \leq B$.

On the other hand, suppose there is a row $\jmath \in \mathcal{J}$ such that $\sum_{i \in C_\jmath} g_i \leq B$. If we choose $x^* = C_\jmath$ we obtain a feasible attack that blocks every defender's solution since all elements that cover $\jmath$ have been interdicted. □

This proposition allow us to determine in polynomial time (just a single scan of the matrix A) if the instance is trivial, and if so, identifies the best attack as

$$\jmath = \arg\min_{j \in \mathcal{J}} \sum_{i \in C_j} g_i \quad \Longrightarrow \quad x^* = C_\jmath$$

The classical approaches to solve the Set Covering Problem perform a pre-processing phase to reduce the instance by finding *essential* columns, that certainly belong to an optimal solution, removing *dominated* columns and rows, and fixing columns based on reduced costs or other mathematical properties (Beasley, 1987). We here discuss whether these concepts can be extended to the SCIP.

An essential column is a column that is included in every defender's feasible solution. In practice, a column $e \in \mathcal{I}$ is essential if and only if there is some row $j \in \mathcal{J}$ such that $C_j = \{e\}$. Since essential columns must necessarily be included in the defender's solution, they are unlikely to exist in a reasonable SCIP instance. In fact, if there is an essential column, the attacker could just interdict it obtaining an attack which satisfies the condition of Proposition 9. Only if all essential columns have interdiction costs strictly bigger than the attack budget, we can apply the reduction by imposing their presence in the defender's optimal solution and, therefore, erasing the covered rows.

A column $d \in \mathcal{I}$ is dominated by a subset of columns $D \subseteq \mathcal{I}$ when all rows that d covers (and possibly other ones) can be covered by D at smaller or equal cost:

$$R_d \subseteq \bigcup_{i \in D} R_i \wedge c_d \geq \sum_{i \in D} c_i$$

In this case, d can be discarded. A row $f \in \mathcal{J}$ is dominated by a row $f' \in \mathcal{J}$ when covering f' implies that also f is covered:

$$C_{f'} \subseteq C_f$$

In this case, removing f from $\mathcal{J}$ does not change the optimal solutions.

Column and row domination often have a waterfall effect, since removing columns could generate new dominated rows and removing rows could generate new dominated columns. Unfortunately, in the SCIP, only dominated rows can be directly removed. In fact, even if column d is dominated by D, the defender might want to use d when some columns of D are interdicted. However, the following properties hold.

Proposition 10. *If each column in $D = \{d_1, \ldots, d_k\}$, taken singularly, dominates column d (i.e., $\{d_1\}$ dominates d, $\{d_2\}$ dominates d, etc.) and $\sum_{l=1}^{k} g_{d_l} + g_d > B$, then d can be erased.*

Proposition 11. *If column d is dominated by subset D, then it is useless to interdict d unless some column in D is also interdicted.*

Properties 10 and 11 will be exploited in Section 6.5.2 implementing them inside of an exact algorithm for the SCIP.

Chapter 6

An Exact Method for Hard Interdiction Problems

In this chapter, we discuss the state of the art exact approaches to solve HIP. In particular we consider the classical Single level reformulation*, which is well-suited for SIP, based on a* Benders Decomposition*; and the* Covering Decomposition *approach. Then we present two new algorithmic frameworks: the first is an adaptation of the classical* Single level reformulation *for HIP; the second is a generalisation of the* Covering Decomposition *which is able to produce a super-optimal bound during the computation. Moreover, we propose three heuristic methods to generate diverse and good-quality defender's solutions that allow to speed up the computation times of the presented algorithms. Experimental results, conducted on the SPIP and SCIP, show that the generalised covering decomposition and the adaptation of the single level reformulation have similar performances, with the former being slightly more efficient than the latter. Moreover, we compare the three heuristic generation methods with the ones from the literature showing major improvements in the running time of the algorithms.*

Since binary interdiction problems are very hard to solve, most exact approaches in the literature follow a specific algorithmic pattern: consider a restricted set of defender's solutions and produce a feasible attack which depends on such restricted set; compute the best defender's solution for the given attack; insert such solution in the restricted set; reiterate until a termination condition is satisfied.

In this work, we discuss the state of the art methods for solving binary interdiction problems which are based on *Single level reformulations* and *Covering Decompositions*. Then, we adapt the former for the Hard interdiction specific case and generalise the latter in order to provide better results. Moreover, since all these methods are based on considering a restricted set of defender's solution, we introduce three new methods to generate a multitude of suitable solutions to decrease the number of iterations of the algorithms.

6.1 The state-of-the-art general framework

The general algorithmic structure used in the literature is based on the iterative refinement of a lower and an upper bound on the optimal solution. In particular, it corresponds the state-of-the-art approaches proposed by Israeli and Wood (2002) (Algorithm 1E) or Lozano and Smith (2017) (Algorithm 1 for the attacker's problem).

6.1.1 Restricted Response Relaxation

Since the scheme previously mentioned involved the computation of a super-optimal bound, we need to find a way to compute one. A typical approach to compute a super-optimal bound is to solve a relaxation of the original problem (e.g., the continuous relaxation, a Lagrangean relaxation, a surrogate relaxation, etc.). In this case, we will use a relaxation of problem (21) presented in detail in Lozano and Smith (2017), but previously exploited also in other works, such as Israeli and Wood (2002).

The *restricted response relaxation (RRR)* consists in considering only a subset $Y' \subseteq Y$ of the defender's feasible solutions and computing an attack $x \in X_B$ that implies the maximum value in the best solution of the defender *restricted* to Y'.

$$RRR(Y'): \quad \max_{x \in X_B} \min_{y \in Y'(x)} c(y) \tag{26}$$

Notice that RRR(Y') is, in fact, a relaxation of HIP because the feasible region X_B remains unaltered, but the objective value for every solution $x \in X_B$ becomes not smaller than the original value:

$$\min_{y \in Y'(x)} c(y) \geq \min_{y \in Y(x)} c(y) = c(x)$$

since it is trivial to prove that $Y'(x) \subseteq Y(x)$. It is important to point out that, while the attacker's problem is relaxed, the defender's one is more constrained, since its solution space gets smaller.

Proposition 12. *Given $Y_1 \subseteq Y_2 \subseteq Y$, $RRR(Y_1) \geq RRR(Y_2)$*

Proposition 12 holds because $\forall x \in X_B : Y_1(x) \subseteq Y_2(x)$. Its practical meaning is that bigger subsets Y' produce better upper bounds. Ultimately, we will search for some $Y' \subseteq Y$ such that $RRR(Y')$ produces a solution of value $c(x^*)$.

Before discussing in more detail, in Section 6.1.3, the ways proposed in the literature to solve RRR(Y'), we present a general algorithmic framework that is currently adopted by the state-of-the-art approaches and exploits the capability of solving the restricted response relaxation.

6.1.2 A general algorithmic structure

Building on the concepts introduced above, the algorithm manages a restricted set of defender's solutions and iteratively refines a lower and an upper bound on $c(x^*)$ by computing a new attack based on the restricted set of defender's moves to possibly update the incumbent, and one or more responses to enlarge the current set Y'.

```
1  $y^* = \arg\min_{y \in Y} c(y)$
2  $LB = c(y^*)$
3  $UB = +\infty$
4  $\bar{x} = \emptyset$
5  $Y' = \{y^*\}$
6  while $LB < UB$ do
7      $\hat{x} = \arg\max_{x \in X_B} \min_{y \in Y'(x)} c(y)$ // Solve RRR($Y'$)
8      $UB = \min_{y \in Y'(x)} c(y)$
9      $\hat{y} = \arg\min_{y \in Y(\hat{x})} c(y)$ // Solve the defender's problem for $\hat{x}$
10     if $c(\hat{y}) > LB$ then // Update the incumbent
11         $\bar{x} = \hat{x}$
12         $LB = c(\hat{y})$
13     end
14     generate a heuristic subset $Y'' \subseteq Y \setminus Y'$
15     $Y' = Y' \cup \{\hat{y}\} \cup Y''$
16     $Y' = Y' \setminus \{y \in Y' | c(y) > UB\}$
17 end
18 return $\bar{x}$
```

Algorithm 9: Algorithm for the exact computation of HIP.

More in detail, Algorithm 9 starts by solving the defender's problem and, thus, obtaining an initial value for the lower bound LB. The initial value of the upper bound UB is set to infinity, the best known attack $\bar{x}$ is empty and the set Y' contains only the optimal defender's solution. Then, the algorithm tightens the lower and the upper bound until they match. First, it solves the relaxation RRR(Y'), which provides a feasible attack $\hat{x}$ and an upper bound. Thanks to Proposition 3 in Lozano and Smith (2017), the upper bound never increases, and therefore there is no need to check if it improves the previous one. The true objective value of $\hat{x}$ is then evaluated by computing (with any ad-hoc algorithm) the optimal defender's solution ($\hat{y}$) in response to $\hat{x}$. If the objective value of $\hat{x}$, that is $c(\hat{y})$, is greater than the current lower bound, it means that the attack $\hat{x}$ is more effective than the incumbent and, therefore, update $\bar{x}$. Then, we extend the restricted defender's solution set Y' with $\hat{y}$. Along with it, we might also want to add a suitably generated set Y'' of heuristic solutions (we will discuss this later in Section 6.4). At last, we erase from Y' any solution whose cost exceeds UB, since such solutions will never affect the optimum of RRR(Y'), and therefore there is no need to keep them in Y' (as showed in Lozano and Smith (2017)). The termination of this algorithm is guaranteed by the fact that each iteration generates a different attack $\hat{x}$ and X_B is finite (see Proposition 4 in Lozano and Smith (2017)).

The versions of Algorithm 9 proposed in the literature mainly differ in terms of:

- the update of the best solution found up to that moment;
- the method used to solve RRR(Y') at line 7;
- the procedures to generate heuristic solutions at line 14.

6.1.3 Solving the Restricted Response Relaxation

Solving problem RRR(Y') allows to find a new attack and an upper bound on the optimum of the HIP. It would be convenient to have a method that could compute RRR(Y') without explicitly considering each solution in Y'. However, this approach appears challenging, and the only attempt in the literature to do so (by Malik et al. (1989)) had a conceptual flaw stemming precisely from not evaluating individual solutions but rather their aggregation as a whole.

Different ways to perform this task have been proposed in the literature. We here describe the main ones.

Single-level reformulation The literature on SIP (Israeli and Wood, 2002; Cappanera and Scaparra, 2011; Lozano and Smith, 2017; Furini et al., 2019) often adopts the following single-level reformulation:

$$\max \quad \theta \tag{27.1}$$

$$\text{s.t.}$$

$$\theta \leq c(y) + \sum_{i \in y} d_i \cdot x_i \qquad y \in Y' \tag{27.2}$$

$$\sum_{i \in I} g_i \cdot x_i \leq B \tag{27.3}$$

$$x_i \in \{0, 1\} \qquad i \in I \tag{27.4}$$

where, for each $i \in I$, d_i is the increment, due to the interdiction of i, in the objective value of any solution y such that $i \in y$. The problem becomes equivalent to a Hard interdiction problem when $\forall i \in I : d_i > c(x^*)$. We can therefore apply this formulation to Hard interdiction by setting a sufficiently large value of d_i (a so called *big M*). In particular, Fischetti et al. (2019) show how to compute smart values for d_i if the problem has the *downward monotonicity property*.

Furthermore, Lozano and Smith (2017) observe that it is possible to strengthen the coefficients and rewrite constraints (27.2) as

$$\theta \leq c(y) + \sum_{i \in y} \min\{d_i, UB + \varepsilon - c(y)\} \cdot x_i \qquad y \in Y' \tag{28}$$

where ε is a very small positive constant.

They also point out that, if an attack $x \in X_B$ is such that $c(y) + \sum_{i \in y} d_i \cdot x_i > UB$ for every solution $y \in Y'$, then formulation (27) cannot improve the upper bound. For the sake of efficiency, attack x can be directly returned without solving it to optimality. To keep this into account, the authors propose to add the following constraint:

$$\theta \leq UB + \varepsilon \tag{29}$$

Covering decomposition Israeli and Wood (2002) propose a family of *super-valid inequalities* to strengthen up formulation (27). These inequalities require to know a lower bound LB on the optimum and hold for all solutions $y \in Y'$ with $c(y) \le LB$. They are of the form

$$\sum_{i \in y} x_i \ge 1 \qquad y \in Y' : c(y) \le LB \tag{30}$$

If $c(y) \le LB$ for all $y \in Y'$, these inequalities are satisfied by the attacks that disrupt all known defender's solutions. The best response $\hat{y}$ to such an attack can be added to Y' to proceed with the next iteration. The authors propose an exact and an approximate way to exploit these properties. The idea is to replace the original formulation with the following parametric *covering decomposition*:

$$\min \quad \sum_{i \in I} g_i \cdot x_i \tag{31.1}$$

$$\text{s.t.}$$

$$\sum_{i \in y} x_i \ge 1 \qquad y \in Y' : c(y) \le \beta \tag{31.2}$$

$$x_i \in \{0,1\} \qquad i \in I \tag{31.3}$$

which represents the search for an attack that disrupts every defender's solution with a cost not larger than β in the current restricted set. Any feasible solution of (31) with a value $\le B$, be it optimal or heuristic, identifies such an attack, and allows to proceed with the general iterative algorithm. On the other hand, if the optimum of formulation (31) is $> B$, no feasible attack is able to cover (that is, to interdict) all the required solutions. Then, some solution $y^+ \in Y'$, with $c(y^+) \le \beta$, is not interdicted by x^*, implying that $y^+ \in Y(x^*)$ and $c(x^*) \le c(y^+) \le \beta$. This terminates the process, thanks to the following theorem enunciated and proved by Israeli and Wood (2002) (Theorem 6).

Theorem 3. *If x^+ is the optimal solution of formulation* (31) *for some value β and $\sum_{i \in I} g_i \cdot x_i^+ > B$, then β is a valid upper bound on the optimum of HIP.*

Israeli and Wood (2002) propose an exact and an approximated version of Algorithm 9. The exact one sets $\beta = LB$ in each iteration, so that, when the algorithm terminates, lower bound and upper bound coincide and HIP is solved to optimality. The approximated version sets $\beta = LB + \varepsilon$, with $\varepsilon > 0$ a tolerance parameter; when it terminates, the best feasible attack has an absolute approximation guarantee equal to ε. The main drawback of both versions is that they produce a valid upper bound only at the end of the computation, which is non-polynomial. Moreover, the approximated approach requires to guess a value of ε corresponding to a reasonable computational time. The next section proposes a remedy for these disadvantages.

6.2 A minor improvement on the single level reformulation

Now, we present our adaptation of the single level reformulation to the Hard interdiction case. We apply the improved constraints (28), (29) as well as the super-valid inequalities (30), but also the following possible strengthening. Since Hard interdiction is equivalent to setting all coefficients $d_i = +\infty$, we should use coefficient $UB + \varepsilon - c(y)$ in all constraints (28). Notice that Algorithm 9 at line 16 discards all solutions in Y' that exceed the upper bound UB, implying that the objective value of all solutions in Y' do not exceed UB. That is also true when a finite upper bound is unknown, and $UB = +\infty$. In our application of constraints (28), we will replace UB with

$$c_{\max}(Y') := \max_{y \in Y'} c(y)$$

In fact, solving the formulation with this setting produces useful information even when $c_{\max}(Y')$ is strictly smaller than $c(x^*)$, and therefore not a proper upper bound. Suppose, in fact, that $c(x^*) > c_{\max}(Y') \Leftrightarrow \min_{y \in Y(x^*)} c(y) > \max_{y \in Y'} c(y)$. Since $Y(x^*)$ and Y' have disjoint cost ranges, $Y(x^*) \cap Y' = \emptyset$, implying that x^* interdicts all solutions in Y'. Consequently, when the optimum of the formulation exceeds $c_{\max}(Y')$, we know that the upper bound is still infinite, but the attack x obtained can be used to generate a response that enlarges Y'.

So, our reformulation for RRR(Y') will assume the following form.

$$\max \quad \theta \tag{32.1}$$

s.t.

$$\theta \leq c(y) + (c_{\max}(Y') + \varepsilon - c(y)) \sum_{i \in y} x_i \qquad y \in Y' \tag{32.2}$$

$$\theta \leq c_{\max}(Y') + \varepsilon \tag{32.3}$$

$$\sum_{i \in y} x_i \geq 1 \qquad y \in Y' : c(y) \leq LB \tag{32.4}$$

$$\sum_{i \in I} g_i \cdot x_i \leq B \tag{32.5}$$

$$x_i \in \{0, 1\} \qquad i \in I \tag{32.6}$$

The inclusion of the super-valid inequalities in formulation (32) is discretionary, as they only serve to tighten the formulation. In fact, in Section 6.5.1, we show that their use in the SPIP results in worse outcomes than when they are not used.

6.3 A new approach for the Restricted Response Relaxation

We here discuss an alternative approach to solve RRR(Y'). The basic idea is to apply a covering decomposition, but include in the restricted set Y' also defender's solutions y with a cost $c(y)$ strictly larger than the optimum and sort them in non-decreasing cost order. Then, the objective value of any attack $x \in X_B$ will be the cost of the non-interdicted solution of smallest index. Notice that this is true only for Hard interdiction problems, not for other kinds of interdiction. The following proposition exploits this property to solve RRR(Y').

Proposition 13. *Given $Y' = \{y_1, y_2, \dots, y_k\} \subseteq Y$ such that $c(y_1) \leq c(y_2) \leq \dots \leq c(y_k)$ (without loss of generality), for any attack $x \in X_B$ there exists $q \in \mathbb{N}$ with $q \leq k$ such that x interdicts the first q solutions of Y'. If $q < k$, solution y_{q+1} is not interdicted by x and $c(y_{q+1})$ is the objective value of x in RRR(Y').*

The previous proposition shows that solving RRR(Y') amounts either to showing that Y' is fully interdictable with the current budget or to identifying a specific solution in Y'. As discussed above, the former case can be easily tested solving formulation (31) on the whole of Y'. For the latter one, we propose the following formulation:

$$\max \sum_{y \in Y'} c(y) \cdot t_y \tag{33.1}$$

s.t.

$$\sum_{y \in Y'} t_y = 1 \tag{33.2}$$

$$t_y \leq 1 - x_i \qquad y \in Y', i \in y \tag{33.3}$$

$$t_y \leq \sum_{i \in y'} x_i \qquad y, y' \in Y' : c(y') < c(y) \tag{33.4}$$

$$\sum_{i \in I} g_i \cdot x_i \leq B \tag{33.5}$$

$$t_y \in \{0,1\} \qquad y \in Y' \tag{33.6}$$

$$x_i \in \{0,1\} \qquad i \in I \tag{33.7}$$

where the binary variables x_i simply represent an attack, assuming value 1 if the element i is interdicted, and 0 otherwise. One of the binary variables t_y assumes value 1 to identify a response y of minimum cost that is not interdicted by attack x; the other ones are equal to 0. The objective value (33.1) is equal to the cost of that solution. Constraint (33.2) imposes a single t_y variable with value 1, forcing the others to 0. Constraints (33.3) ensure that $t_y = 0$ for any solution $y \in Y'$ interdicted by an attacked element i. Constraints (33.4) ensure that $t_y = 1$ only if y is an uninterdicted solution of minimum cost: they consider every pair of solutions $y, y' \in Y'$ such that $c(y') < c(y)$ and force $t_y = 0$ whenever y' is not interdicted ($\sum_{i \in y'} x_i = 0$). Constraint (33.5) imposes to respect the interdiction budget.

Notice that this formulation is incorrect when some attack $x \in X_B$ can interdict all solutions in Y', since it will select a solution $y^+ \in Y'$ and assign $t_{y^+} = 1$. Since $y^+ \in Y'$

we know that even though $c(y^+) \leq c_{\max}(Y') < c(x^*)$ and it will not produce a valid upper bound. However, this situation is ruled out in advance.

Formulation (33) uses more variables and constraints than formulation (32), but presents smaller coefficients, and this could be related to tighter integrality gaps. With respect to the covering decomposition, it allows (and, actually, encourages) the inclusion in Y' of solutions with a cost larger than the current lower bound. If prematurely terminated, it provides both an upper and a lower bound, instead of simply a lower bound, without the need to guess the upper bound. Consequently, it does not pose *a priori* the alternative between converging to the optimum (Algorithm 2 in Israeli and Wood (2002)) or getting a solution with a predefined absolute approximation (Algorithm 2E).

The approach described above is related to the formulation presented by Scaparra and Church (2008) for a fortification-interdiction p-median problem. In this problem, given a set of locations that currently host facilities, first the defender selects a subset to immunise, then the attacker selects another subset in order to maximise the overall service cost. The approach enumerates the feasible attacks, assuming that their number is tractable. Then, it sorts by increasing objective values the ones that range between a suitable lower and upper bound (computed with an interpolation technique and a sequence of auxiliary Set Covering problems). Finally, it finds the best fortification pattern solving a suitable MILP formulation on the considered attacks. The first enhancement introduced by our approach is to consider a restricted subset of the lower level solutions, instead of generating them exhaustively. The second one is that auxiliary Set Covering problems are used to directly determine the optimum, instead of narrowing the range between the lower and upper bound and, therefore, the size of the formulation.

Formulation (33) actually requires heavy and slow computations. Nevertheless, we can exploit its structure (the imposition of a single variable equal to 1) to decompose it into a sequence of easier sub-problems. The idea is to select a suitable solution $y \in Y'$ and set $t_y = 1$ and $t_{y'} = 0$ for all the other ones. Then we try to compute a feasible attack x which fits these conditions on the t variables. We denote this problem as *Restricted Response Covering Decomposition (RRCD(Y', y))*. By imposing such conditions on formulation (33) we obtain

$$
\begin{aligned}
\max \quad & c(y) & \\
\text{s.t.} \quad & & \\
x_i &= 0 & i \in y \\
\sum_{i \in y'} x_i &\geq 1 & y' \in Y' : c(y') < c(y) \\
\sum_{i \in I} g_i \cdot x_i &\leq B & \\
x_i &\in \{0,1\} & i \in I
\end{aligned}
$$

which is a pure feasibility problem, because the objective function is a constant.

Therefore, we convert the budget constraint into an objective function as follows

$$
\begin{aligned}
\min \quad & \sum_{i\in I} g_i \cdot x_i & & \quad (34.1)\\
\text{s.t.} \quad & & & \\
& x_i = 0 & i \in y & \quad (34.2)\\
& \sum_{i\in y'} x_i \geq 1 & y' \in Y' : c(y') < c(y) & \quad (34.3)\\
& x_i \in \{0,1\} & i \in I & \quad (34.4)
\end{aligned}
$$

obtaining a formulation similar to the covering decomposition (31), but tightened by the additional constraint (34.2) that the attack cannot include the elements that interdict the selected response.

If such an attack does not exist or has an interdiction cost larger than the budget, it is clear that we should consider a solution y of non-larger cost. If, on the contrary, we can find (heuristically or exactly) a solution $\tilde{x}$ such that $\sum_{i\in I} g_i \cdot \tilde{x}_i \leq B$, the values of the $\tilde{x}$ variables with the corresponding ones for the t variables provide a feasible solution for formulation (33).

```
1  $T = Y'$
2  $\hat{x} = \emptyset$
3  loop
4      $\tilde{y} = \arg\max_{y\in T} c(y)$
5      solve RRR$(Y', \tilde{y})$ with formulation (34) obtaining $\tilde{x}$
6      if $\sum_{i\in I} g_i \cdot \tilde{x}_i \leq B$ then
7          $\hat{x} = \tilde{x}$
8          exit loop
9      end
10     $T = T \setminus \{\tilde{y}\}$
11 end
12 return $\hat{x}$
```

Algorithm 10: Procedure for the computation of RRR(Y') via sequential set covering computations.

Algorithm 10 allows to compute an optimal solution of formulation (33). Set T includes the candidate uninterdictable solutions of Y' (at the beginning, $T = Y'$), while $\hat{x}$ (initialised as an empty set) will return at the end the optimal attack solution of RRR(Y'). Of course, we assume that RRR(Y') is feasible: Y' should contain at least a solution that can be interdicted within the available budget. The algorithm iteratively selects the candidate solution $\tilde{y} \in T$ of maximum objective value and solves RRCD$(Y', \tilde{y})$ with formulation (34), obtaining the optimal solution $\tilde{x}$. If $\tilde{x}$ respects the budget, it is saved in $\hat{x}$ and returned, because it is the optimal solution of RRR(Y'). The corresponding upper bound is $c(\tilde{y})$. Otherwise, $\tilde{y}$ is removed from T and the next candidate solution is considered. Instead of moving sequentially from the solution of maximum cost, one could apply more refined schemes (such as dichotomic search).

However, in practice the optimal solution is nearly always very close to the maximum one, and therefore, the naive strategy only requires very few iterations. Intuitively, this derives from the fact that Y' is a restricted set that contains only solutions whose cost respects the current upper bound. Of course, it is also possible to solve RRCD$(Y', \tilde{y})$ heuristically at line 5, instead of finding the optimum, provided that the objective of the heuristic solution is $\leq B$. In fact, the real issue is to find a feasible attack covering all solutions cheaper than $\tilde{y}$. The algorithm is correct, as it just applies an implicit branching on t_y in formulation (33). Its overall complexity corresponds to solving RRCD(Y', y), which is equivalent to a Set Covering problem, at most $|Y'|$ times, since $|T|$ starts from $|Y'|$ and decreases by one at each iteration. Since Y' contains only solutions y with $c(y) \leq UB$ and the final value of $c(\tilde{y})$ is the new upper bound, the solutions that are removed from T during the computation will be removed also from Y' in line 16 of Algorithm 9.

We now want to emphasize that if Y' was populated exclusively by solutions with an objective value less than or equal to the LB, this method would be entirely equivalent to the covering decomposition, as an UB would never be identified. Moreover, since the procedure relies on solving Set Covering problems, this new approach can be seen as a *generalised* version of the covering decomposition.

Reduction procedures The following new theorem shows that logical properties of the relation between attacks and responses can be exploited to remove further solutions from Y', thus reducing the computational effort.

Theorem 4. *Given $\check{y} \in Y'$, if RRCD$(Y', \check{y})$ does not admit any feasible solution (i.e., formulation* (34) *cannot produce an attack with objective $\leq B$), then $\check{y}$ can be permanently erased from Y' without affecting the optimal value of RRR(Y').*

Proof. The statement is true when $c(\check{y}) > UB$, because removing solutions whose objective is greater than the upper bound does not affect the optimum, therefore we focus on the opposite case: $c(\check{y}) \leq UB$. By hypothesis, the optimum of RRCD$(Y', \check{y})$ is $> B$, meaning that no feasible attack is able to interdict all solutions with objective $< c(\check{y}) \leq UB$ without interdicting also $\check{y}$. On the other hand, we know that it is possible to interdict with a single feasible attack $\hat{x} \in X_B$ all solutions in Y' with objective $< UB$ (that is, a larger set of solutions), otherwise we could improve the UB using Theorem 3. Hence, RRCD$(Y', \check{y})$ violates the budget because it asks for a solution which does not interdict $\check{y}$. In other words, the only way to interdict all solutions with objective $< c(\check{y})$ without violating the budget B is to interdict also $\check{y}$. This means that the presence of $\check{y}$ inside Y' does not affect RRR(Y'), allowing to erase the said solution from Y'. □

Theorem 4 allows to delete solutions from Y' as soon as they fail to produce a feasible attack using formulation (34). Consequently, Algorithm 10 can directly use Y' instead of the auxiliary set T.

6.4 Diversification strategies

So far, we have assumed to include in the restricted set Y' only the best response $\hat{y}$ generated at each step. When $c(\hat{y}) > LB$, this mechanism updates the incumbent and the lower bound, so that Y' is populated only by solutions with cost $\leq LB$. Consequently, RRR(Y') does not provide an upper bound as long as all responses generated are interdictable, that is, until the optimality condition is satisfied. This basically reduces the approach to the exact covering decomposition of Israeli and Wood (2002), without exploiting the additional results presented above.

It is therefore of paramount importance to generate and introduce in Y' also some heuristic solutions, as suggested by line 14 of Algorithm 9. The literature provides different techniques to obtain good heuristic solutions for specific interdiction problems. Considering the SPIP, for example, Israeli and Wood (2002) propose a *local search* technique to generate several $s-t$ paths from a single shortest path tree. Lozano and Smith (2017) describe an adaptation of the *pulse algorithm* to generate several good quality $s-t$ paths with limitations on the number of generated solutions that insist on the same arcs. They also discuss a more general approach, based on the optimal response of the defender to a random attack x. Leitner et al. (2023) propose a constructive procedure to generate heuristic solutions (that, in their case, are attacks).

Some of these techniques are *ad-hoc* and not easy to adapt to general problems. Many are thought for SIP, and do not work as well on the Hard version. In the former case, in fact, even elements interdicted in the optimal attack could still be part of an optimal response, whereas in the latter it is a good defence strategy to diversify as much as possible the generated solutions, because that will make it harder for the attacker to interdict all of them. An approach aware of this point is more likely to pursue the aim better than simply reacting with the optimal response to the last attack, and should speed up the convergence to the optimum. In the following, we propose a few general *diversification mechanisms* to produce sufficiently different heuristic solutions for HIP.

We assume that a defender's solution is a subset of elements of a finite *ground set* J ($\forall y \in Y : y \subseteq J$) and that the objective function is additive: $c(y) = \sum_{j \in y} c_j$, where c_j is the cost of the single element $j \in J$. We denote as $Y^\top \subseteq Y$ the set of all solutions generated and added to $Y^{'}$ up to the current point in the computation (including also the removed ones). For each interdictable element $i \in I$, we maintain a counter of the solutions $y \in Y^\top$ such that $i \in y$:

$$\forall i \in I \qquad o_i := |\{y \in Y^\top | i \in y\}| \tag{35}$$

and we combine these counters to define the auxiliary objective function:

$$\forall y \in Y \qquad o(y) := \sum_{i \in y} \frac{o_i}{g_i} \tag{36}$$

Then, we consider a bi-objective defender's problem, searching for efficient trade-offs between the cost of the solution and the repeated use of basic elements of the ground set.

$$\begin{aligned} \min \quad & c(y) \\ \min \quad & o(y) \\ \text{s.t.} \quad & y \in Y(\hat{x}) \end{aligned}$$

where $\hat{x}$ is the last generated attack.

On the one hand, in fact, solutions with a cost larger than the upper bound are useless. On the other hand, solutions blocked by elements with a high number of occurrences in $Y^\top$ are easier to interdict focusing on those elements. The problem aims to avoid two complementary failures: generating an expensive solution and obtaining again the optimal one $\hat{y}$. In addition, our experience suggests that also the computation (at line 9 of Algorithm 9) of the optimal response $\hat{y}$ with respect to $c(y)$ benefits from considering $o(y)$ as a secondary objective in case of ties. Notice that, while the original objective function remains the same throughout the computation, the auxiliary one changes adaptively.

The next subsections describe three heuristic procedures to generate good and diversified solutions.

6.4.1 Lexicographic bi-objective

The first procedure considers a lexicographic bi-objective version of the defender's problem, where the diversification index $o(y)$ is the main objective, while the cost $c(y)$ is secondary.

$$\begin{aligned} \min \quad & c(y) \\ \text{s.t.} \quad & y \in Y(\hat{x}) \\ & o(y) = \min_{y' \in Y(\hat{x})} o(y') \end{aligned}$$

For some problems, such as the *SPIP*, this can be directly tackled with a dedicated algorithm. For other problems, such as the *SCIP*, optimising the main objective and then imposing its value as a constraint and optimising the secondary objective can be inefficient. Since the approach is heuristic in principle, we can resort to an approximate reformulation that optimises a linear combination of the two objectives:

$$\begin{aligned} \min \quad & c'(y) = c(y)/\gamma + o(y) \\ \text{s.t.} \quad & y \in Y(\hat{x}) \end{aligned}$$

with a sufficiently large coefficient γ. When the interdiction cost is unitary, ($\forall i \in I : g_i = 1$), as often assumed in the literature, function $o(y)$ always has integer values and any $\gamma > \max_{y \in Y(\hat{x})} c(y)$ (e.g., $\gamma = |J| \cdot \max_{j \in J} c_j + 1$) guarantees that minimising c' solves the lexicographic bi-objective problem. For integral values of g_i, it is possible to rescale $o(y)$ in order to obtain the same property.

Since both objectives are additive, the reformulated problem can be solved with the same algorithm used for the original one, replacing the original cost function with $c' : J \rightarrow \mathbb{R}^+$:

$$c'_j = \begin{cases} c_j/\gamma + o_j/g_j & j \in I \\ c_j/\gamma & j \notin I \end{cases}$$

In fact:

$$c'(y) = c(y)/\gamma + o(y) = \sum_{\substack{j\in y:\\ j\notin I}} (c_j/\gamma) + \sum_{\substack{j\in y:\\ j\in I}} (c_j/\gamma) + \sum_{\substack{j\in y:\\ j\in I}} (o_j/g_j) =$$
$$= \sum_{\substack{j\in y:\\ j\notin I}} (c_j/\gamma) + \sum_{\substack{j\in y:\\ j\in I}} (c_j/\gamma + o_j/g_j) = \sum_{j\in y} c'_j$$

At each iteration of Algorithm 9 (line 14), therefore, we compute $y'' = \arg\min_{y\in Y(\hat{x})} c'(y)$ and check its cost $c(y'')$. If $c(y'') \leq UB$, we add y'' to Y'' and increment by one the counters o_i with $i \in y''$; otherwise, y'' is useless and we reset $Y^\top = \emptyset$ and $o_i = 0$ for all $i \in I$.

6.4.2 Adaptive convex combination

While diversification is important, focusing too much on it is not necessarily effective. Suitably tuned convex combinations of $o(y)$ and $c(y)$ could be preferable. Our second reformulation solves:

$$\begin{aligned} \min \quad & c^\alpha(y) = (1-\alpha)\cdot c(y) + \alpha \cdot o(y) \\ \text{s.t.} \quad & y \in Y(\hat{x}) \end{aligned}$$

where coefficient $\alpha \in [0,1]$ tunes the weight of the two objectives. In fact, when $\alpha = 0$, $c^0(y) = c(y)$, whereas $\alpha = 1$ corresponds to $c^1(y) = o(y)$. As in the previous case, additivity allows to solve the problem with the same algorithm used for the original one.

Notice that, as α decreases and the focus moves from o to c, the optimal solution with respect to $c^\alpha(y)$ has monotonically non-increasing costs. Therefore, Algorithm 9 starts with $\alpha = 1$ to maximise diversification. At line 14, given the last generated attack $\hat{x}$, it computes solution $y'' = \arg\min_{y\in Y(\hat{x})} c^\alpha(y)$ and performs the following adaptive update of α:

- if $c(y'') > UB$, update $\alpha = \alpha \cdot 0.9$ to reduce the cost;
- if $c(y'') \leq UB$ and $y'' = \hat{y}$, update $\alpha = \alpha \cdot 1.1$ to increase the diversification;
- if $c(y'') \leq UB$ and $y'' \neq \hat{y}$, insert y'' into Y''.

The rationale is that finding a bad solution indicates an excessive focus on diversification and finding the cheapest solution indicates an excessive focus on cost, whereas finding a good solution different from the optimal one is a positive outcome, that suggests to confirm the current coefficient Notice that, performing a single update per iteration, the auxiliary function $o(y)$ changes at every application, providing a moving target for the desired value of α.

6.4.3 Dichotomic search convex combination

Our third diversification mechanism is based on the following reformulation

$$\begin{aligned} \min \quad & o(y) \\ \text{s.t.} \quad & y \in Y(\hat{x}) \\ & c(y) \leq UB \end{aligned}$$

which aims to generate the most different solution from the previously generated ones, but with a cost $\leq UB$. In general, the additional constraint does not allow to solve this problem with the same algorithm as the basic one. However, thanks to the monotonicity of $c(y^\alpha)$ with respect to α, it is possible to obtain a heuristic approximation by minimising c^α with α set to the maximum value (α^{best}) for which the cost respects the upper bound.

$$\begin{aligned} \max \quad & \alpha \\ \text{s.t.} \quad & \alpha \in [0,1] \\ & y^\alpha = \arg \min_{y \in Y(\hat{x})} c^\alpha(y) \\ & c(y^\alpha) \leq UB \end{aligned}$$

This approach is heuristic because it finds the best *supported* solution, instead of the best one overall (Ehrgott, 2005).

```
1  $\alpha^{\text{lb}} = 0$
2  $\alpha^{\text{ub}} = 1$
3  $Y'' = \emptyset$
4  **while** $\alpha^{\text{ub}} - \alpha^{\text{lb}} > \varepsilon$ **do**
5      $\alpha = (\alpha^{\text{lb}} + \alpha^{\text{ub}})/2$
6      $y'' = \arg\min_{y \in Y(\hat{x})} c^\alpha(y)$
7      **if** $c(y'') \leq UB$ **then**
8          $Y'' = Y'' \cup \{y''\}$
9          update $o_i$ for all $i \in I$
10         $\alpha^{\text{lb}} = \alpha$
11     **else**
12         $\alpha^{\text{ub}} = \alpha$
13     **end**
14 **end**
15 **return** $Y''$
```

Algorithm 11: Dichotomic search for the largest value of α that respects the upper bound.

Algorithm 11 implements line 14 in Algorithm 9 computing α^{best} with a dichotomic search. This has the possible additional advantage to generate several more or less diversified solutions, instead of the single one generated by the first two mechanisms. The procedure starts by initialising the range for α^{best}, between $\alpha^{\text{lb}} = 0$ and $\alpha^{\text{ub}} = 1$ and the set Y'' of all generated solutions to an empty set. Then, it iterates, setting α to the middle point of the current range and computing the solution y'' that minimises $c^{\alpha}(y)$ and is not interdicted by the attack $\hat{x}$. If its cost is $c(y'') \leq UB$, we add it to Y'', update the counters o_i and remove the lower half of the range, because by monotonicity $\alpha^{\text{best}} > \alpha$. Otherwise, we remove the upper half of the range, because we cannot produce a solution with $c(y'') \leq UB$ and, therefore, $\alpha^{\text{best}} < \alpha$. The algorithm terminates when the range between the two bounds falls below a small constant ε, so that the total number of iterations is $r^{\max} := \log_2(1/\varepsilon)$.

6.4.4 Preliminary observations about the diversification methods

We here discuss the possible pros and cons of the diversification methods presented in this work and also the two previously described from the literature.

Analysing the computational complexity, we can notice that the Lexicographic method is a very simple method which quickly produces good solutions by performing a single execution of the defender's problem algorithm; the Adaptive method also requires a single execution of such algorithm; the Dichotomic method intensifies the effort on finding a good solution by performing a dichotomic search instead of a single attempt; the Local Search method by Israeli and Wood (2002) (similarly to the first two methods) only requires a computation of Dijkstra's algorithm and a little extra effort; the Pulse method by Lozano and Smith (2017), after computing a shortest path tree (via Dijkstra's algorithm), performs a DFS to explore the solutions tree, stopping after a given tractable number of solutions is found.

Analysing the effectiveness in diversifying the solutions, the Adaptive method should produce better solutions than the Lexicographic method because the former considers a convex combination which includes also the objective of the latter (α close to 1) and adaptively tunes α; the Dichotomic method should furthermore improve the results by searching for the best value for α; the Local Search method produces various solutions which share a sub-path with the optimal solution, and therefore fail to effectively diversify; the Pulse method has a similar limitation as the Local Search method since during a DFS exploration of the solution space, a lot of solutions with the same initial sub-path are produced.

Moreover, our methods are designed to work on a variety of interdiction problems whereas the two from the literature are specific for the SPIP.

6.5 Experimental results

In this section we present the computational results obtained by using Algorithm 9 to solve SPIP and SCIP with the techniques presented in the previous sections. The purpose of this experimental section is to determine which of the techniques described in the previous sections performs best for the SPIP and the SCIP. In particular, we apply Algorithm 9 and we solve the restricted response relaxation RRR(Y') either with the single-level reformulation (32), the classical covering decomposition (31) or with the generalised covering decomposition of Algorithm 10 and formulation (34). We will refer to the first approach as SLR, the second as CD, and the final one as GCD. Then, we will combine these approaches with the diversification mechanisms described in Section 6.4.1 (LEX), Section 6.4.2 (ADA) and Section 6.4.3 (DIC), alongside with no diversification mechanism at all (None).

For simplicity reasons, we only consider instances of the two problems with unitary interdiction costs. This is a common choice in the literature, for which the budget B actually becomes the cardinality of the attacks on the network.

When some instance is not solved to optimality before the time limit of 1 hour, we measure the quality of the returned solution. To do so, we estimate the total optimality gap with (19) as previously done for the exact algorithm for the WSSP.

The choice of the value ε (used in DIC) for both problems is $\varepsilon = 1/(|J| \cdot |Y^\top|)$ where $|J|$ is the maximum number of elements in any solution and $|Y^\top|$ is an upper bound on the maximum counter of any element in any solution. This choice is motivated by the fact that, doing so, when the original objective $c(y)$ is integral, the precision of the dichotomic search is enough to allow α to be so small that the auxiliary objective $o(y)$ becomes secondary, obtaining a lexicographic bi-objective problem. Moreover, the maximum number of iterations becomes $r^{\max} = \log_2(|J|) + \log_2(|Y^\top|)$.

6.5.1 Shortest Path Interdiction Problem

We consider the instances generated by Lozano and Smith (2017), discarding the cost-increment due to arc interdiction, because we consider the Hard version of the SPIP, instead of the soft version. These instances have the same structure as those in Israeli and Wood (2002) and Cappanera and Scaparra (2011). They consist in directed graphs $G = (N, A)$ with two special nodes $s, t \in N$. All other nodes are disposed in a 2D-grid fashion, with the same number of rows and columns denoted as $n \in \{10, 20, 30, 40, 50, 60\}$. The source node s has no in-going arcs, but has out-going arcs to all nodes in the first column. Conversely, the destination node t has no out-going arcs, but has in-going arcs from all nodes in the last column. Every node of the grid has one arc going to the node above and the node below in the same column, to the node on the right in the same row and to the node above and the node below the node on the right. The nodes in the first and the last row and in those in the last column make exception, because they lack some adjacent nodes above, below or on the right. Figure 6.1 illustrates a very simple instance with $n = 4$.

The costs of the arcs are random integer numbers with a uniform distribution between 1 and ρ with $\rho \in \{10, 100\}$. We consider the attack budgets $B \in \{5, 6, 7, 8, 9\}$ and

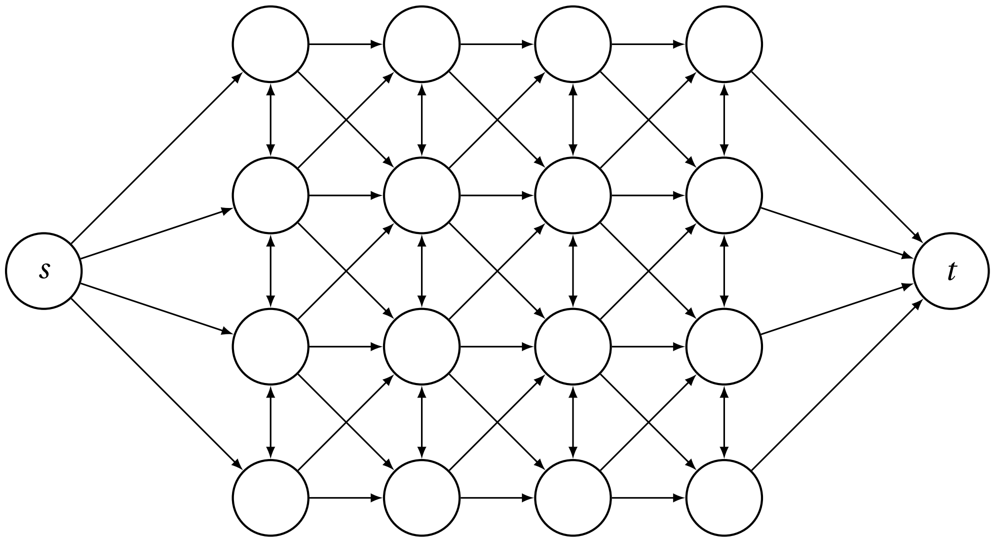

Figure 6.1: Structure of a 4 × 4 directed grid instance according to the Israeli and Wood (2002) model.

10 instances for each size, randomization parameter and attack budget obtaining $6 \cdot 2 \cdot 5 \cdot 10 = 600$ instances.

The first experiment measures the impact of the *super-valid inequalities (SVI)* (30) using formulation (32) in the context of HIP. We compare the time consumption of SLR with and without the SVI over all benchmark instances, considering the diversification mechanisms introduced in Section 6.4.

The second experiment is to determine which approach is better for the SPIP. We confront the best performing version of SLR from the previous experiment with GCD considering all diversification mechanism introduced in Section 6.4 over all benchmark instances. Moreover, we compare the results of the best performing setting with the diversification mechanisms presented in Israeli and Wood (2002) (*Local Search: LS*) and Lozano and Smith (2017) (*Pulse*) over a subset of "easier" instances consisting in the ones with attack budgets $B \in \{5, 6, 7\}$.

Table 6.1 associates each row to the benchmark instances with the same parameters B and n. There are three macro-columns: the first reports the values B and n, the second is associated with the SLR approach without SVI and the third to the SLR approach with SVI. The second and third macro-columns are composed by 4 columns, each of which associated with a diversification mechanism. Each entry of the table exhibits the average computation time to solve the associated instances with the associated approach. The last row, instead of the average computing time, displays the sum of the computation times of all instances.

		SLR				SLR+SVI			
B	n	None	LEX	ADA	DIC	None	LEX	ADA	DIC
5	10	0.14	0.15	0.17	0.11	**0.09**	0.16	0.15	0.11
	20	**0.33**	0.50	0.37	0.37	0.43	0.63	0.43	0.44
	30	**0.67**	0.85	0.80	0.98	0.97	1.71	0.78	1.07
	40	4.10	12.64	**2.47**	3.94	4.83	16.56	3.60	4.49
	50	6.29	4.44	**3.04**	6.29	4.35	10.37	3.35	5.23
	60	6.21	10.93	**5.22**	8.74	16.71	23.53	10.58	9.84
6	10	0.23	0.36	0.30	**0.20**	**0.20**	0.34	0.31	0.21
	20	1.07	1.53	0.99	**0.77**	1.55	2.32	1.14	1.03
	30	**1.28**	2.13	1.50	1.64	2.27	3.60	2.00	2.04
	40	29.66	50.57	15.29	**13.60**	53.54	98.26	23.17	23.11
	50	9.28	32.78	**7.87**	10.74	19.92	48.79	13.83	11.77
	60	17.46	40.34	**10.80**	16.91	31.83	58.35	13.61	18.83
7	10	0.51	0.92	0.66	0.44	0.61	0.91	0.66	**0.42**
	20	2.68	3.59	2.18	**1.45**	4.53	4.96	2.71	1.83
	30	3.82	5.59	3.29	**3.19**	6.97	7.12	6.22	3.92
	40	62.47	68.66	**18.55**	25.81	161.62	118.48	55.39	37.85
	50	40.63	146.86	38.97	**33.10**	90.73	265.24	48.33	49.97
	60	114.67	147.72	54.51	**53.29**	210.67	227.94	74.48	81.40
8	10	1.41	2.17	1.38	**0.90**	1.68	2.50	1.51	0.96
	20	6.15	8.49	4.98	**3.30**	14.53	10.11	6.40	3.73
	30	12.63	16.62	7.72	**6.91**	21.20	23.35	11.57	9.36
	40	79.42	86.55	**62.00**	64.83	228.07	142.04	81.16	137.61
	50	230.48	445.63	113.33	**106.59**	388.61	462.08	180.71	156.33
	60	317.90	293.95	**123.51**	129.23	696.64	499.51	229.01	229.91
9	10	3.55	4.00	2.41	1.42	4.24	4.16	2.50	**1.34**
	20	19.54	25.66	14.81	9.25	34.72	32.56	18.38	**9.23**
	30	35.27	40.60	19.52	**14.27**	73.82	53.74	27.90	18.38
	40	201.98	340.78	**98.39**	109.98	573.51	423.35	201.37	213.69
	50	573.31	687.85	**381.22**	395.66	1012.46	822.96	588.19	519.36
	60	747.40	816.73	337.03	**211.25**	1169.83	1023.95	631.73	575.69
sum		50611	65992	26665	**24703**	96622	87792	44823	42583

Table 6.1: Comparisons between the time consumption of SLR with and without SVI.

We can observe that approaches adopting the SVI rarely produce better results than those that do not. Moreover, the two columns that produce the better values are associated with SLR using ADA and DIC.

Given this, Table 6.2 shows the same information regarding GCD and compares it with the two columns that produced the best results in the previous table.

		GCD				SLR	
B	n	None	LEX	ADA	DIC	ADA	DIC
5	10	0.09	**0.08**	0.09	0.09	0.17	0.11
	20	0.29	0.38	**0.28**	0.36	0.37	0.37
	30	0.50	0.68	**0.44**	0.72	0.80	0.98
	40	3.47	6.28	**1.97**	3.17	2.47	3.94
	50	3.20	4.82	**2.00**	3.72	3.04	6.29
	60	9.89	10.30	**4.48**	7.65	5.22	8.74
6	10	0.19	0.17	0.17	**0.15**	0.30	0.20
	20	1.07	0.90	**0.66**	0.69	0.99	0.77
	30	1.64	1.49	**1.28**	1.41	1.50	1.64
	40	26.60	41.37	14.77	**12.28**	15.29	13.60
	50	14.62	18.34	**7.04**	10.25	7.87	10.74
	60	25.19	23.81	**9.24**	13.00	10.80	16.91
7	10	0.51	0.43	0.37	**0.30**	0.66	0.44
	20	3.15	2.51	1.60	**1.44**	2.18	1.45
	30	3.73	3.49	3.15	**2.77**	3.29	3.19
	40	89.74	58.83	20.74	23.18	**18.55**	25.81
	50	58.00	112.48	**22.96**	32.73	38.97	33.10
	60	141.79	103.22	43.60	**42.52**	54.51	53.29
8	10	1.35	1.04	0.77	**0.58**	1.38	0.90
	20	7.38	5.41	3.34	**2.83**	4.98	3.30
	30	10.84	8.73	5.99	**5.78**	7.72	6.91
	40	105.83	61.95	49.71	**36.97**	62.00	64.83
	50	309.60	263.26	**73.17**	90.16	113.33	106.59
	60	483.08	220.84	128.48	**104.38**	123.51	129.23
9	10	3.33	1.83	1.32	**0.88**	2.41	1.42
	20	18.52	13.33	9.73	**7.74**	14.81	9.25
	30	36.92	23.30	15.41	**12.68**	19.52	14.27
	40	315.25	281.59	94.17	**80.66**	98.39	109.98
	50	640.56	520.36	408.60	393.70	**381.22**	395.66
	60	766.06	556.78	298.01	281.42	337.03	**211.25**
sum		61648	46960	24471	**23484**	26665	24703

Table 6.2: Comparisons between the time consumption of SLR and GCD.

It is possible to notice that, for almost every row, the GCD approach produces better results than SLR. In particular, the diversification mechanisms ADA and DIC often perform best but the last two rows show slightly smaller gaps for the SLR approaches. Moreover, the same GCD approaches also produce the minimum sum of the time consumptions over all instances.

Another interesting aspect to consider is the number of iterations of Algorithm 9 performed by adopting GCD and SLR with all the presented diversification mechanisms. Table 6.3 has 3 macro-columns: the first has two columns which specify the parameters B and n of the considered instances; the second is associated to the GCD approach and has 4 columns dedicated to the diversification mechanisms; the third macro-column is identical to the previous one but for the SLR approach. For each row, we exhibit the average number of iterations among the considered instances (the ones with the same values of B and n), performed by all the methods. Moreover, the last row, instead of the average, for each column exhibits the sum of the iterations over all the instances. We highlight in bold the minimum value for each row.

		GCD				SLR			
B	n	None	LEX	ADA	DIC	None	LEX	ADA	DIC
5	10	48	34	39	21	48	35	40	**20**
	20	79	67	61	33	71	64	59	**32**
	30	89	82	69	**37**	97	84	80	43
	40	169	169	109	**71**	172	182	118	77
	50	154	154	104	**59**	165	137	118	75
	60	246	213	144	**79**	214	213	149	90
6	10	72	52	50	**25**	75	52	52	26
	20	128	106	88	49	127	108	97	**48**
	30	165	131	116	**59**	147	137	119	65
	40	356	327	223	**128**	355	348	225	139
	50	306	260	178	**107**	273	287	185	110
	60	367	325	212	**121**	348	370	212	137
7	10	113	75	73	**36**	122	76	75	37
	20	223	150	130	**67**	213	153	133	**67**
	30	247	182	164	**85**	266	201	164	89
	40	639	465	309	**181**	622	531	303	202
	50	537	471	263	189	477	536	311	**180**
	60	774	595	386	**232**	748	641	431	264
8	10	186	103	96	**48**	197	106	102	51
	20	333	207	174	**86**	332	212	182	96
	30	359	273	214	**119**	439	301	234	134
	40	749	527	379	**219**	784	590	494	320
	50	956	720	422	**276**	955	796	459	295
	60	1446	863	556	**337**	1301	861	570	370
9	10	277	117	115	**49**	303	121	115	51
	20	535	293	251	**128**	546	296	258	130
	30	620	367	308	**152**	698	405	315	165
	40	1240	851	512	**308**	1277	894	571	357
	50	1676	1017	661	**416**	1695	1012	669	483
	60	1870	1123	712	**424**	1948	1184	815	425
sum		299118	206373	142338	**82864**	300304	218630	153087	91533

Table 6.3: Number of iterations of Algorithm 9 performed by each of the presented approaches.

As we expected, for both GCD and SLR, when no diversification mechanism is used, the number of iteration is higher, followed by the LEX mechanism which slightly decreases the number of iterations. The ADA mechanism further improves the result by halving the number of iterations with respect to None. At last, DIC displays the minimum number of iterations among all methods obtaining a smaller order of magnitude with respect to None. The differences between GCD and SLR are not very marked but, fixing a diversification method, GCD often seems to produce fewer iterations.

Now, we are interested in the instances that remain unsolved after 1 hour of computation for at least one of the considered approaches. Table 6.4 associates each row to one of the 8 unsolved instances and each column to one of the considered approaches. The entries display the gap produced at the end of the 1 hour expiration time and, when the instance is solved to optimality, a "-" is exhibited.

			GCD				SLR			
n	ρ	B	None	LEX	ADA	DIC	None	LEX	ADA	DIC
50	100	8	-	-	-	-	-	2.41%	-	-
50	10	8	-	-	-	-	-	1.75%	-	-
40	10	9	-	-	-	-	-	1.11%	-	-
50	100	9	∞	4.52%	3.37%	3.11%	∞	4.24%	2.99%	2.74%
50	10	9	∞	1.74%	-	-	-	0.86%	-	-
60	100	9	∞	-	-	-	∞	0.22%	-	-
60	100	9	∞	-	-	-	∞	1.25%	-	-
60	10	9	∞	0.75%	-	-	∞	-	-	-
			∞	0.88%	0.42%	0.39%	∞	1.48%	0.37%	**0.34%**

Table 6.4: Comparisons between the gaps produced by SLR and GCD over the unsolved instances.

For these challenging instances, the best gaps are produced by the SLR approach with DIC that solves to optimality 7 out of 8 instances and produces a gap of 2.74% for the remaining one. That said, similar results are produced by SLR with ADA but also by GCD with ADA and DIC.

Comparison with the state of the art Now, we want to compare the new approaches (GCD and SLR with DIC) with the ones in the literature. As far as we know, the best performing algorithms for the SPIP are presented by Israeli and Wood (2002) and Lozano and Smith (2017). Both of them present a *Benders decomposition*-based approach. Israeli and Wood (2002) also present a *Covering Decomposition.*

To the best of our knowledge, there are no computational results, in the literature, about the hard version of the SPIP, and therefore we performed additional experiments also for the state-of-the-art approaches. The *Benders decomposition*-based approaches are designed for the soft case, therefore we adapt them for the hard case as described in Section 6.2 with the improvement there described. Obviously, the LS and Pulse mechanisms have been re-implemented. In particular, the Pulse algorithm has been translated from the original Java code[1] to C++.

Table 6.5 has a structure similar to the previous tables. Here, the second macro-column is dedicated to the state-of-the-art methods that we have identified in Israeli and Wood (2002) (CD+LS, SLR+LS) and Lozano and Smith (2017) (SLR+Pulse). The third macro-column is dedicated to the GCD method with the LS, Pulse and DIC heuristic generation mechanisms. We only reported the results for instances with $B \in \{5, 6, 7\}$ because these methods require too much time for harder instances.

		state of the art			GCD		
B	n	SLR+LS	CD+LS	SLR+Pulse	LS	Pulse	DIC
5	10	0.16	**0.05**	0.75	0.31	10.64	0.09
	20	0.67	**0.15**	3.67	1.93	43.73	0.36
	30	1.90	**0.54**	13.80	7.78	91.72	0.72
	40	8.71	**2.86**	63.01	35.70	178.19	3.17
	50	25.28	12.92	99.62	61.30	188.63	**3.72**
	60	36.85	10.26	160.14	133.76	264.98	**7.65**
6	10	0.32	**0.09**	1.34	0.61	24.54	0.15
	20	2.15	**0.38**	7.52	5.74	104.91	0.69
	30	4.60	**1.00**	28.54	19.60	178.76	1.41
	40	28.51	12.95	119.39	80.14	254.30	**12.28**
	50	53.64	24.11	158.57	138.03	243.95	**10.25**
	60	82.65	36.45	248.78	204.06	406.56	**13.00**
7	10	0.75	**0.17**	2.79	1.23	44.30	0.30
	20	5.95	**1.07**	16.98	12.47	206.24	1.44
	30	12.29	2.87	60.64	50.56	291.73	**2.77**
	40	60.10	36.44	185.28	128.97	300.57	**23.18**
	50	89.91	51.19	233.55	186.55	304.67	**32.73**
	60	133.40	85.98	266.46	340.26	409.64	**42.52**
sum		10956.95	5589.70	33416.59	28179.72	70961.15	**3128.55**

Table 6.5: Comparisons between the state-of-the-art methods and the new ones over the easier instances.

The results show that, once again, the Covering Decomposition method by Israeli and Wood (2002) with the Local Search heuristic generation, performs best for the smaller instances. On the other hand, the GCD approach with DIC outperforms all the others for the bigger instances and handles larger values of B effectively as it scales up.

[1]Available online as "Supplementary Material" at `https://doi.org/10.1287/ijoc.2016.0721` (Lozano and Smith, 2017).

The same results are displayed in a graphical form in Figure 6.2.

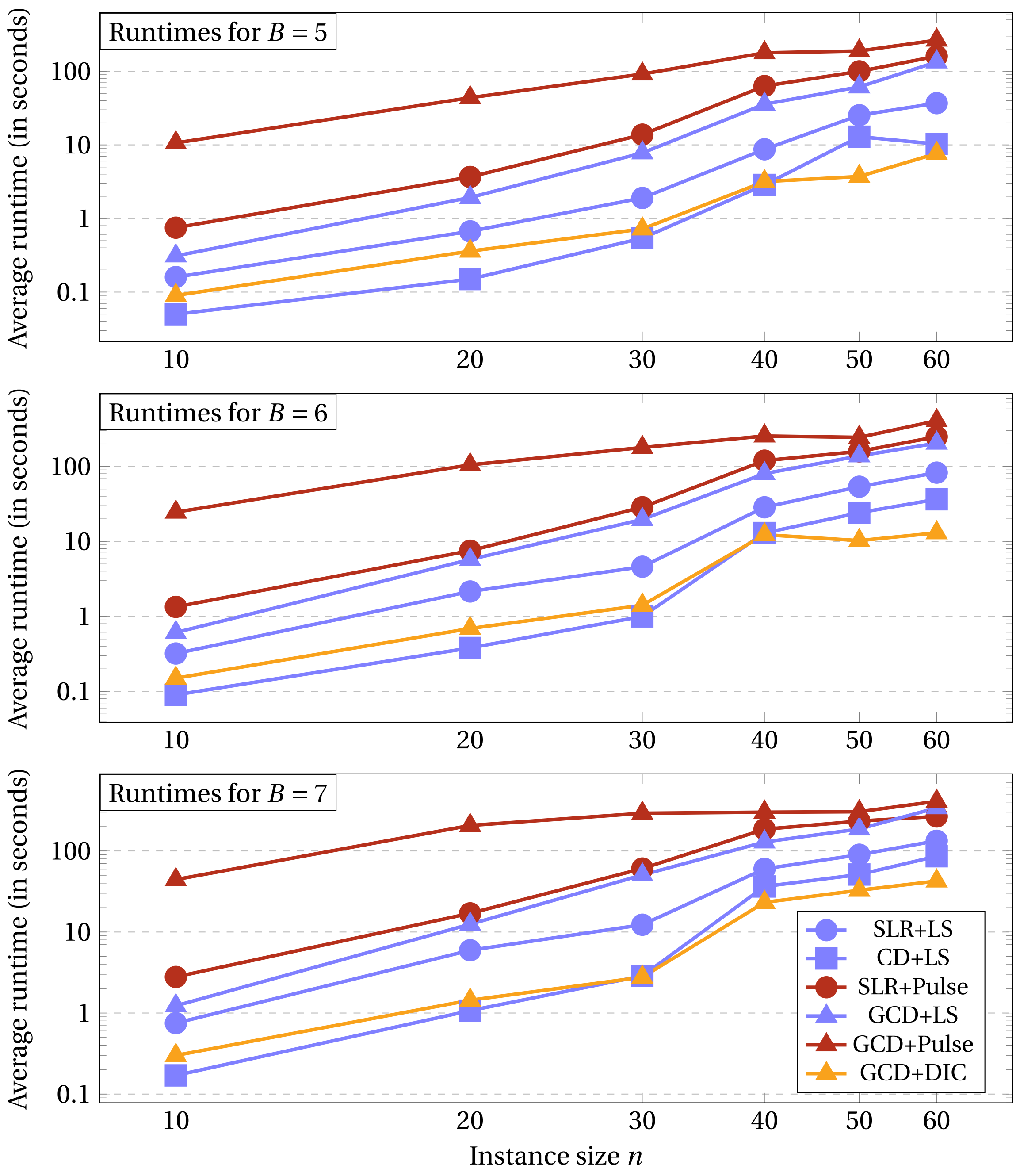

Figure 6.2: Time consumptions of the considered approaches in logarithmic scale.

6.5.2 Set Covering Interdiction Problem

For the SCIP we introduce a new benchmark of randomly generated instances. They are represented by binary square matrices $A \in \{0,1\}^{n\times n}$ with $n \in \{100, 150, 200\}$, so that $\mathcal{I} = \mathcal{J} = \{1,\dots,n\}$. The density parameter

$$\delta := \frac{\sum_{i=1}^n \sum_{j=1}^n A_{ij}}{n^2} = \frac{\text{number of 1 entries}}{\text{total number of entries}}$$

ranges in $\{0.1, 0.2, 0.3, 0.4\}$. The covering costs c_i for each column $i \in \mathcal{I}$ are generated uniformly at random between 1 and 100, whereas the interdiction costs $g_i = 1$ for each $i \in \mathcal{I}$ are unitary. For each size n and density δ we consider 5 instances and the interdiction budget $B \in \{5,6,7,8,9\}$. Overall, the number of considered instances in the benchmark is $3\cdot 4\cdot 5\cdot 5 = 300$.

To strengthen formulation (34) for the SCIP, we introduce the following cuts, based on Proposition 11 and generated during a pre-processing phase.

$$\text{If } D \subseteq \mathcal{I} : D \text{ dominates } d \quad \Longrightarrow \quad x_d \le \sum_{i\in D} x_i \tag{43}$$

Moreover, we also apply Proposition 10 to erase all columns dominated by a good number of other columns. Unfortunately, preliminary results show that the effectiveness of these enhancements is negligible for the instances considered. In fact, we do not observe any significant change in execution times compared to not using them at all.

The first experiment aims to narrow down the most promising approaches by using only a subset of "easy" instances: we only consider the instances with $\delta \in \{0.2, 0.3, 0.4\}$ and $B \in \{5,6,7\}$.

Table 6.6 shows the results for GCD. Each row is associated to a group of 5 instances with the same values of B, δ, n and the first three columns identify such parameters; the following four columns are associated to the diversification mechanisms presented in Section 6.4. Each entry displays the average computing time to solve the associated instances with the associated approach. We also report the sum of the computation times over all instances with the same budget B. For each line we highlight in bold the minimum values among the 4 considered approaches.

B	δ	n	None	LEX	ADA	DIC
5	0.2	100	9.96	14.09	8.11	**7.43**
		150	22.62	31.60	**18.77**	20.78
		200	59.23	74.76	**40.20**	41.82
	0.3	100	3.26	4.53	2.95	**2.61**
		150	9.65	11.24	8.45	**6.13**
		200	24.86	26.83	22.98	**15.81**
	0.4	100	1.98	2.18	1.56	**1.13**
		150	4.81	5.69	4.35	**2.64**
		200	18.08	20.11	15.65	**9.34**
	sum all		772.29	955.08	615.07	**538.44**
6	0.2	100	33.64	50.39	**22.50**	23.47
		150	89.42	123.08	**60.71**	70.75
		200	199.04	282.48	**120.55**	141.21
	0.3	100	6.77	10.56	5.96	**4.93**
		150	20.81	27.73	15.94	**13.47**
		200	47.66	55.39	39.69	**29.44**
	0.4	100	3.31	4.40	2.72	**2.03**
		150	8.38	11.51	7.19	**4.56**
		200	33.90	39.66	26.36	**16.59**
	sum all		2214.66	3025.99	**1508.12**	1532.22
7	0.2	100	106.88	179.36	**73.85**	74.55
		150	309.60	475.68	**209.09**	247.58
		200	728.94	1134.79	**423.36**	481.72
	0.3	100	15.21	26.22	13.87	**12.59**
		150	46.37	69.00	34.02	**29.12**
		200	86.91	119.62	71.42	**54.00**
	0.4	100	5.69	8.48	4.83	**3.55**
		150	14.30	21.43	12.30	**8.12**
		200	60.22	70.06	44.32	**28.38**
	sum all		6870.58	10523.19	**4435.29**	4698.06

Table 6.6: Average time consumption to solve SCIP using GCD with all diversification mechanisms on the "easy" instances.

Table 6.7 reports the same information as Table 6.6 with respect to the SLR approach.

B	δ	n	None	LEX	ADA	DIC
5	0.2	100	**9.24**	44.53	25.71	18.2
		150	**21.71**	104.52	71.25	46.8
		200	**54.24**	200.21	137.08	85.17
	0.3	100	**3.40**	8.53	5.49	4.37
		150	**8.41**	22.44	14.68	8.96
		200	24.39	34.9	28.09	**18.59**
	0.4	100	1.87	3.03	2.1	**1.44**
		150	4.88	7.19	5.05	**3.08**
		200	17.7	22.87	15.71	**9.92**
	sum all		**729.20**	2241.13	1525.79	982.59
6	0.2	100	**30.50**	216.79	111.1	66.95
		150	**74.49**	501.04	282.73	157.07
		200	**162.04**	946.29	442.08	237.66
	0.3	100	**6.31**	32.19	16.51	11.59
		150	**19.41**	80.33	42.85	25.02
		200	49.05	103.07	65.58	**42.59**
	0.4	100	3.08	8.76	5.27	**3.04**
		150	8.51	18.17	10.2	**5.88**
		200	29.85	49.62	32.14	**18.75**
	sum all		**1916.11**	9781.29	5042.25	2842.71
7	0.2	100	**98.27**	846.02	330.21	185.91
		150	**272.06**	2026.19	902.17	440.48
		200	**518.99**	3136.11	1574.39	657.24
	0.3	100	**14.34**	101.01	49.93	29.71
		150	**48.11**	221.86	110.34	57.83
		200	**79.46**	263.48	143.58	83.55
	0.4	100	**5.8**	20.55	10.58	6.5
		150	13.81	40.7	21.63	**12.24**
		200	57.35	90.73	55.46	**34.24**
	sum all		**5540.99**	33733.30	15991.49	7538.47

Table 6.7: Average time consumption to solve SCIP using SLR with all diversification mechanisms on the “easy” instances.

From Tables 6.6 and 6.7, we can infer that the most promising approaches for GCD are ADA and DIC whereas for SLR are None and DIC. Therefore, the next experiment confronts these 4 approaches over all benchmark instances.

Table 6.8 shares a similar structure with the previous tables except for the fact that we aggregate all instances with the same values of B and n.

		GCD		SLR	
B	n	ADA	DIC	None	DIC
5	100	**18.34**	22.29	20.35	62.66
	150	**82.80**	122.16	87.64	219.60
	200	183.24	312.73	**163.98**	472.94
6	100	**90.53**	114.52	95.33	229.69
	150	475.13	667.31	**459.87**	863.47
	200	882.48	948.36	**872.87**	975.78
7	100	**415.82**	545.28	474.97	780.19
	150	**964.23**	973.81	983.90	1028.39
	200	**1035.08**	1047.26	1064.09	1094.90
8	100	**921.35**	921.96	991.61	1026.65
	150	**1091.17**	1108.82	1197.21	1248.13
	200	**1313.22**	1398.70	1410.28	1471.45
9	100	**1057.73**	1057.77	1207.63	1183.91
	150	**1457.58**	1547.60	1638.33	1706.15
	200	**1829.71**	1856.76	1849.82	1871.06
sum all		**236368**	252907	250358	284700

Table 6.8: Average time consumption to solve SCIP using the 4 promising approaches identified by Tables 6.6 and 6.7 over all benchmark instances.

The results of Table 6.8 show that the four approaches have similar performances but the GCD with ADA often produces the smallest average time consumption as well as the smallest sum.

The same results are displayed in a graphical form in Figure 6.3, aggregating the instances by dimension n, to show the trend of the runtime as the budget B increases.

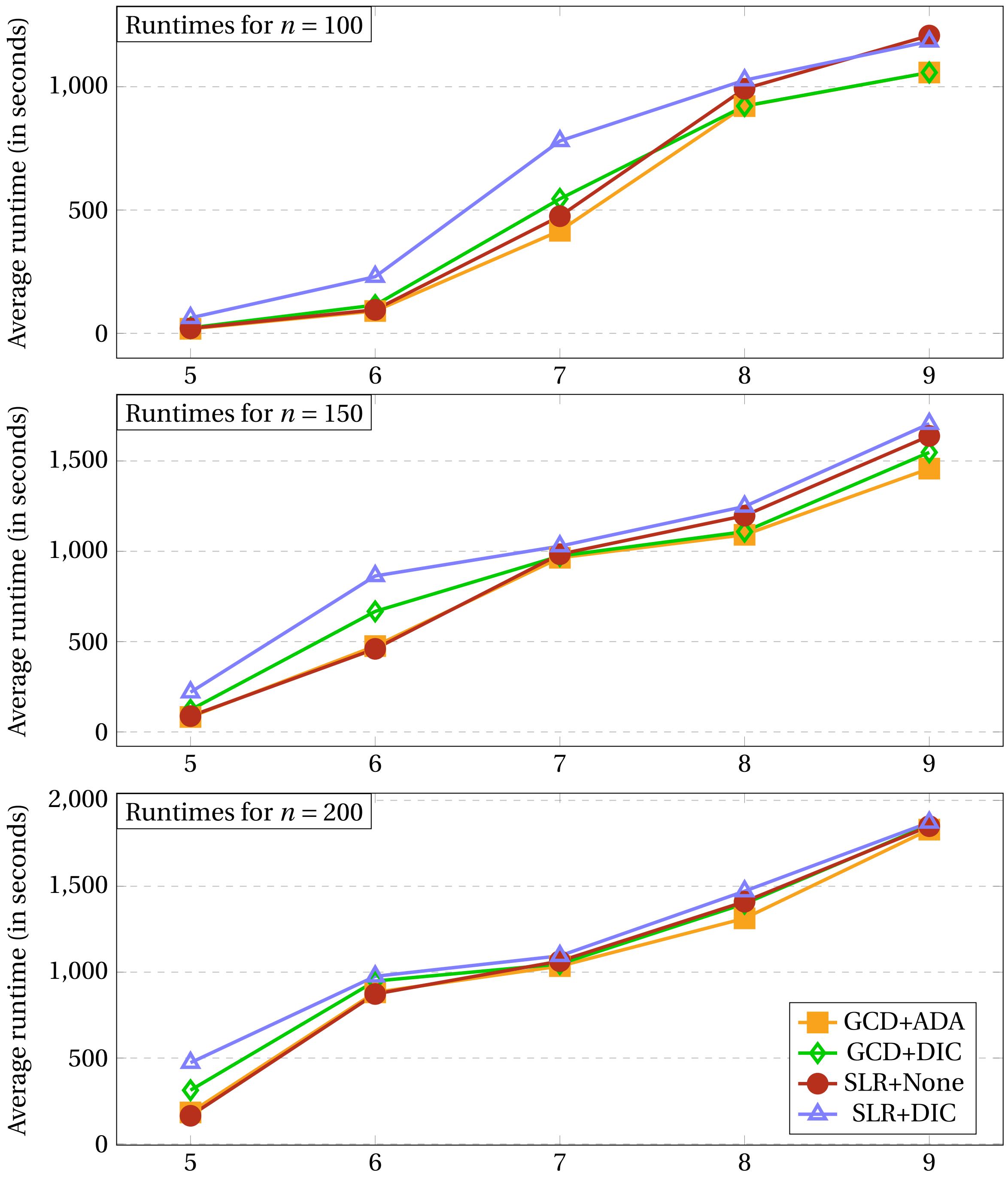

Figure 6.3: Time consumption of the considered approaches, as B increases.

The number of iterations of Algorithm 9 performed by the selected approaches are reported in Table 6.9. The latter shares the same structure with Table 6.8 but, instead of the average time consumption, exhibits the average number of iterations.

		GCD		SLR	
B	*n*	ADA	DIC	None	DIC
5	100	317.35	**188.30**	436.45	191.10
	150	532.00	328.90	712.35	**327.25**
	200	698.40	455.45	923.95	**444.15**
6	100	596.85	332.85	910.50	**332.20**
	150	1059.55	611.25	1601.90	**589.25**
	200	1324.20	637.70	2178.15	**597.75**
7	100	1095.60	593.65	1746.75	**567.20**
	150	1372.60	633.10	2875.35	**600.80**
	200	1242.30	**581.25**	3138.95	590.15
8	100	1441.45	678.45	2867.55	**558.05**
	150	1374.80	654.35	3646.60	**611.95**
	200	1494.85	788.25	4149.65	**703.25**
9	100	1457.95	719.60	3758.35	**533.65**
	150	1651.90	830.40	4715.25	**720.55**
	200	1752.70	858.65	5030.95	**767.35**
sum all		348250	177843	773854	**162693**

Table 6.9: Average number of iterations using the 4 promising approaches, over all benchmark instances.

As expected, of the considered diversification methods, DIC is the one that minimizes the number of iterations, None the one that maximizes it, and ADA between the other two. An interesting aspect is that, although Table 6.8 shows GCD+DIC to be faster than SLR+DIC in terms of time, SLR+DIC requires fewer iterations, suggesting that a single iteration of GCD takes less time than one of SLR.

For what concerns the instances that have not been solved before the 1 hour deadline, we computed the average gaps produced and the number of solved instances. In Table 6.10, each row is associated to the instances which share the same values of B and n. There are 3 macro-columns: the first reports the parameters B and n; the second concerns the produced gaps and the third the number of solved instances. The second macro-column consists of three columns associated with the approaches considered in the previous table, except for SLR with None for which a finite gap is produced only at the end of the computation. Each entry exhibits the average gap produced among all the associated instances by the associated approach. The third macro-column consists of four columns associated with all the approaches considered. Each entry displays the number of instances solved by the associated approach among the associated instances. Recall that every row is associated to 20 instances.

		gap			solved			
		GCD		SLR	GCD		SLR	
B	$\|I\| = \|J\|$	ADA	DIC	DIC	ADA	DIC	None	DIC
5	100	-	-	-	20	20	20	20
	150	-	-	-	20	20	20	20
	200	-	-	-	20	20	20	20
6	100	-	-	-	20	20	20	20
	150	-	0.13%	0.32%	**20**	19	**20**	17
	200	**1.23%**	1.40%	1.59%	16	15	**17**	15
7	100	-	-	0.14%	20	20	20	19
	150	2.81%	**2.45%**	2.69%	15	15	15	15
	200	6.58%	5.06%	**4.98%**	15	15	15	15
8	100	1.75%	**1.09%**	2.15%	**16**	**16**	15	15
	150	7.38%	**5.12%**	6.56%	15	15	15	15
	200	14.41%	**7.68%**	10.60%	15	15	15	15
9	100	5.84%	**2.82%**	7.51%	15	15	15	15
	150	17.70%	**8.06%**	14.10%	15	15	15	14
	200	26.73%	**12.36%**	20.29%	11	11	11	11
average		5.63%	**3.08%**	4.73%				
maximum		123.88%	**52.00%**	100.33%				

Table 6.10: Average gaps and number of instances solved by the four promising approaches over all benchmark instances.

The results show that the minimum average gap (when positive) is associated to the GCD approach with diversification DIC. Moreover, the same approach produces the minimum average gap over all instances, as well as the smallest maximum gap. For what concerns the number of solved instances, we can observe that there are no significant differences between the various approaches since often the number of instances solved to optimality is the same or differs by at most 2.

To better visualize the produced gaps over all the instances, Figure 6.4 provides the Solution Quality Distribution diagram of the considered approaches. The gap values are reported on the horizontal axis whereas the vertical axis describes a fraction of instances as a percentage. For every approach, we report the curve that describes the fraction of instances such that the produced gap is smaller than or equal to a certain gap.

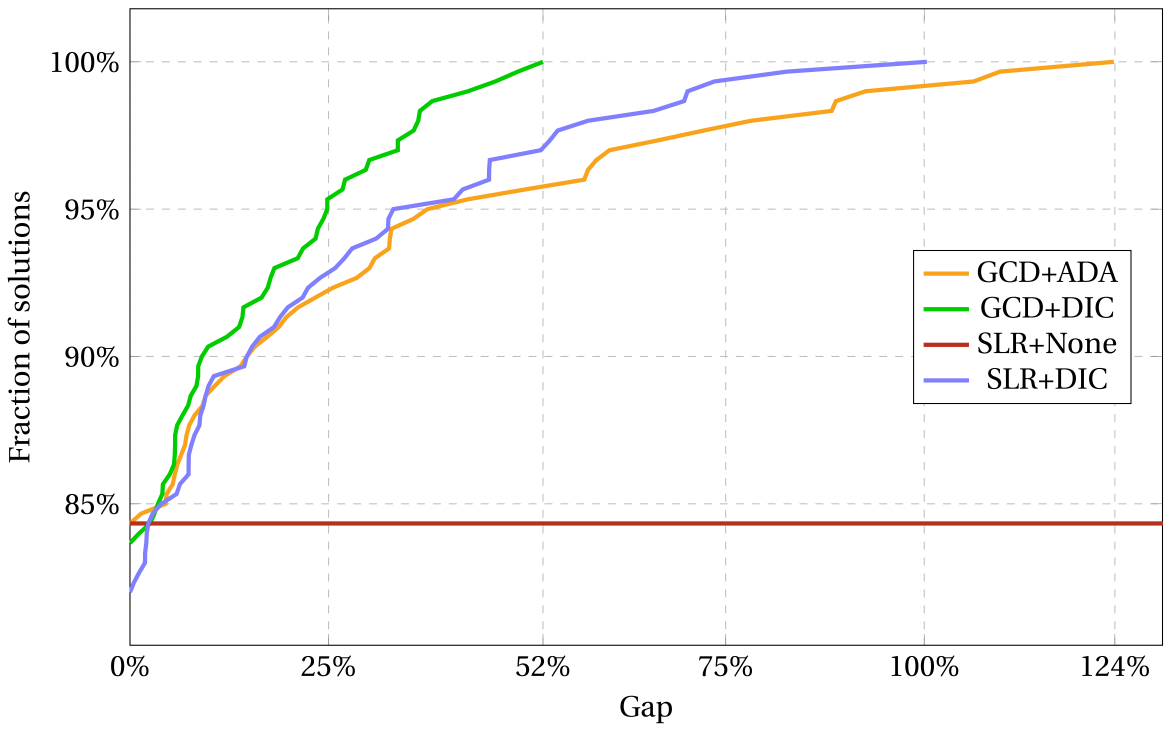

Figure 6.4: SQD for the SCIP over all benchmark instances, using the considered approaches.

Figure 6.4 highlights the positive and negative aspects of all the methods. The methods which solve the highest number of instances to optimality are GCD+ADA and SLR+None with 84.33%. A very close result is achieved by GCD+DIC with 83.67% followed by SLR+DIC with 82%. As the gap grows, GCD+DIC rapidly takes the highest spot, GCD+ADA and SLR+DIC initially perform similarly but eventually the latter pulls ahead of the former, and SLR+None remains constant since it cannot provide any useful upper bound (infinite gap). These results suggest that, even if the time consumptions tend to be similar, among the considered approaches, when the computation is prematurely terminated, GCD+DIC obtains the best results almost halving the worst result with respect to the second best SLR+DIC.

Chapter 7

Resilient Samples for Hard Interdiction Problems

In this chapter, we propose formulations and algorithms to compute Resilient Samples of defender's feasible solutions. A Resilient Sample is a set of solutions which cannot be interdicted with a single attack of given budget. Specifically, we present a technique to compute a small Resilient Sample for the SPIP in polynomial time. Subsequently, we generalise the same technique for a certain class of interdiction problems maintaining the size and polynomial time guarantees. Then, we present a heuristic method for arbitrary interdiction problems that allows to compute a resilient sample whenever the previous approach is not applicable. At last, we present another method for producing a resilient sample for binary fortification and similar problems.

In the previous chapter, we discussed how important and challenging it is to produce a good super-optimal bound for solving HIP. To produce such a bound, it is important to be able to compute a well-diversified and high-quality set of defender solutions. Obviously, one could exploit the diversification mechanisms in Section 6.4, but these methods do not provide any guarantee that the set of solutions will yield a super-optimal bound. For this reason it is important to identify solution sets that cannot be simultaneously interdicted by a single feasible attack: the *Resilient Samples (RS)*. In particular we are interested in RS of small (polynomial) cardinality.

Another reason for the interest in RS is that, from the perspective of the defender, they provide a set of solutions which ensures that at least one option will remains available at all times.

In the following, we will formally define the concept of resilient sample and then present a method that guarantees to compute in polynomial time one of manageable size for the SPIP. We then generalise the method for a broader class of interdiction problems with specific characteristics. For the even more general class of HIPs to which this approach cannot be applied, we propose an alternative method for computing a resilient sample. This last method does not ensure the production of a correct RS and does not guarantee polynomial time; however, it provides a set of solutions that can serve as a useful starting point for exact approaches and may be expanded to form a RS. Finally, we provide a reliable and efficient method for computing a RS for binary fortification problems, that can also be used for any interdiction problem where the solution space of the defender has a knapsack structure.

7.1 Definition

We define the central subject of this chapter.

Definition 3. *Given a budget $B \in \mathbb{R}^+$, a* ***Resilient Sample*** *$Y' \subseteq Y$ is a set of defender's feasible solutions such that for any feasible attack $x \in X_B$, at least one solution in Y' is not interdicted.*

$$\forall x \in X_B : \exists y \in Y' : x \cap y = \emptyset$$

As a consequence of the definition above, we can derive a necessary and sufficient condition for the existence of a RS. Notice that whenever the attacker's budget is insufficient to interdict the whole defender's solution space, then the latter is a RS. On the contrary, whenever the attacker is able to interdict the whole defender's solution space, there cannot be a RS.

Remark 4. *Any interdiction problem admits a RS if and only if the attacker's budget is not sufficient to interdict the whole defender's solution set.*

A RS is a useful object to compute for the following reasons:

- it provides a non-trivial super-optimal bound for the considered interdiction problem (Theorem 3);
- Y' in Algorithm 9 can be initialised with an RS to provide a good diversified restricted set since the beginning;
- from a defensive point of view, it provides a set of "recovery plans" ensuring that at least one of them is still operative after the interdiction.

To check whether a given set Y' is a RS, we can solve formulation (31) with $\beta = +\infty$ and compare the optimal objective value $g(x^*) = \sum_{i \in I} g_i x_i^*$ with the attack budget B.

Remark 5. *Given a subset of defender's feasible solutions $Y' \subseteq Y$ and an attack budget $B \in \mathbb{R}^+$, Y' is a RS if and only if $g(x^*) > B$.*

This fact, even though it does not provide a way to compute a RS, lays the foundations for all the methods we will describe in the following sections.

7.2 Resilient Sample for the Shortest Path Interdiction Problem

In this section, we describe how to compute a RS for the SPIP. We will show that the relationship between the SPIP and the min-cut problem is not just evidenced by the fact that a min-cut provides a supremum for the attack budget (see Section 5.3.1), but also reveals a connection with the max-flow problem that allows us to produce a RS.

To obtain the RS we consider the continuous relaxation of the covering decomposition (31) for the SPIP and analyse its dual formulation. We point out that the dual is similar to a path-decomposition of a feasible flow. Therefore we provide an algorithm to compute such decomposition and demonstrate that the result is a RS for the SPIP.

Given a SPIP instance with a graph $G = (N, A)$, a source node $s \in N$ and a sink node $t \in N$, arc costs $c : A \to \mathbb{R}^+$, interdiction costs $g : A \to \mathbb{R}^+$ and interdiction budget $B \in \mathbb{R}^+$, consider the continuous relaxation of formulation (31):

$$\min \sum_{a \in A} g_a \cdot x_a \tag{44.1}$$

$$\text{s.t.}$$

$$\sum_{a \in y} x_a \geq 1 \qquad y \in Y' \tag{44.2}$$

$$x_a \geq 0 \qquad a \in A \tag{44.3}$$

where we can drop the $x_a \leq 1$ constraints, since it is a minimisation problem.

Now, we consider the dual formulation

$$\max \sum_{y \in Y'} f_y \tag{45.1}$$

$$\text{s.t.}$$

$$\sum_{\substack{y \in Y': \\ a \in y}} f_y \leq g_a \qquad a \in A \tag{45.2}$$

$$f_y \geq 0 \qquad y \in Y' \tag{45.3}$$

which is a path-decomposition based formulation for the maximum flow problem (see Section 3.5 of Ahuja et al. (1993)) with a restricted number of paths and coefficients g as capacities. This is consistent with the fact that formulation (31) is related to the min-cut problem. Moreover, if the maximum $s - t$ flow for the instance (with capacities g) is smaller than or equal to B, we can conclude that the problem is trivial because we can actually disconnect s and t within the given budget and, therefore, no resilient samples can be produced.

Otherwise, let's consider an $s - t$ flow f of value $v(f) > B$. As stated in Theorem 3.5 of Ahuja et al. (1993), any such flow can be decomposed into a polynomial number of paths.

We now present an algorithm to perform this decomposition as follows.

```
Algorithm Decompose($G = (N, A)$, $g : A \to \mathbb{R}^+$, $B \in \mathbb{R}^+$, $f$ feasible flow):
    $Y' := \emptyset$
    $\tilde{f}_y := 0 \quad \forall y \in Y$ // Implicitly
    while $\exists y \in Y : \min_{a \in y}\{f_a\} > 0$ do
        $\tilde{f}_y := \min_{a \in y}\{f_a\}$
        $Y' := Y' \cup \{y\}$
        for $a \in y$ do $f_a := f_a - \tilde{f}_y$
    end
    return $Y', \tilde{f}$
```

Algorithm 12: Algorithm for computing a path-decomposition of a feasible flow f

At the beginning of Algorithm 12, Y' is initialized to $\emptyset$ and the vector $\tilde{f}$ is implicitly (because Y is too big) set to 0 for every entry. Then, in the main cycle (line 4), we look for some $s-t$ path y whose arcs have a positive amount of flow f (we don't specify which path specifically). In particular we look for the arc of minimum flow a and set $\tilde{f}_y := f_a$. Then we decrease the flow of every arc in y by $\tilde{f}_y$ producing a new flow whose value is smaller than the previous by $\tilde{f}_y$. Also notice that $\hat{f}_a = 0$ at the end of the iteration. The algorithm stops when there are no longer $s-t$ paths with positive flow and returns the desired Y' and the vector $\tilde{f}$.

Notice that at the end of this algorithm the remaining flow f might not be null for some arcs in the graph. This is because there might still be cycles which do not contribute to the value of the flow. Moreover, since at each iteration at least one arc is set to 0, the maximum number of iterations of the algorithm is $|A|$ and, since each iteration inserts a single path in Y', also the maximum cardinality of Y' is $|A|$.

The decomposition values $\tilde{f}$ can be used to regain a feasible flow f' with $v(f') = v(f)$ by defining

$$f'_a := \sum_{\substack{y \in Y: \\ a \in y}} \tilde{f}_y = \sum_{\substack{y \in Y': \\ a \in y}} \tilde{f}_y$$

Of course, the second equality holds because $f_y = 0$ for any solution in $Y \setminus Y'$. Since f' is a feasible flow we know that

$$\sum_{\substack{y \in Y': \\ a \in y}} \tilde{f}_y = f'_a \leq g_a \tag{46}$$

Therefore, $\tilde{f}$ is a feasible solution for formulation (45) with value $\sum_{y \in Y'} \tilde{f}_y = v(f') = v(f) > B$.

Now, we can show that the set Y' returned by Algorithm 12 is actually a resilient sample, provided that the input flow f has value $v(f) > B$.

Theorem 5. *Let Y' be the set returned by Algorithm 12, Y' is a* resilient sample *for the SPIP.*

Proof. To prove this theorem, we show that the optimal solution of the covering decomposition (31) has a value $g(x^*)$ bigger than the budget B in accordance with Remark 5. To do so, we prove the following chain of inequalities

$$B < v(f) = \sum_{y \in Y'} \tilde{f}_y \leq (45)^* \leq (44)^* \leq g(x^*)$$

where $(45)^*$ and $(44)^*$ are the optimal values of their respective formulations.

By hypothesis, $v(f) > B$. Since $\tilde{f}$ is a path-decomposition of f which maintains the same value, it follows that $v(f) = \sum_{y \in Y'} \tilde{f}_y$. Given that $\tilde{f}$ is a feasible (possibly non optimal) solution of formulation (45) using Y', we know that $\sum_{y \in Y'} \tilde{f}_y \leq (45)^*$. By weak duality theorem, $(45)^* \leq (44)^*$. At last, since $g(x^*)$ is the optimal solution of (31) and (44) is its relaxation, we conclude that $(57)^* \leq g(x^*)$. □

Thanks to Theorem 5 we know that the optimal solution of formulation (31) using the set Y' produced by Algorithm 12 has objective bigger than B and, therefore, to interdict all paths in Y' it is necessary to pay more than the budget. In other words, Y' is a resilient sample for the SPIP.

Resilient Sample for the Maximum Flow Interdiction Problem Since an $s-t$ path can be treated as a special case of $s-t$ flow, we can use the same technique also for the Maximum Flow Interdiction Problem. In this case we would compute an initial $s-t$ flow with respect to the capacity function g of value $v(f) > B$ and, then, decompose it into paths with Algorithm 12. The objective value of such paths $y \in Y'$ would be equal to the minimum capacity among all the arcs in the path, $c(y) = \min_{a \in y} c_a$. Of course, the quality of such solution is poor, because a single path is usually not able to transport a large quantity of flow. Anyway, one could use these solutions as a starting point to produce better solutions.

7.3 Resilient Sample for defender's problems with integrality property

We now generalize the procedure in the previous section for the following class of interdiction problems.

Definition 4. *A binary interdiction problem is called* Totally Unimodular with Binary Interdictable Variables *(**TUBIV**) if the defender's problem can be modelled with a linear programming formulation such that:*

1. *the coefficients matrix is* totally unimodular *(Hoffman and Kruskal, 1956);*
2. *the right-hand-side coefficients are integers;*
3. *the variables that can be interdicted are binary.*

Notice that these three conditions hold for the special case of the SPIP. The idea is, once again, to produce a resilient set Y' by computing a linear combination of solutions and, then, extracting such solutions one by one together with their positive linear coefficients.

First, we consider a generic linear programming formulation for the defender's problem with the desired requirements. Then we design a non-linear formulation for the problem of finding a RS. Since this formulation has several critical aspects, we will relax it and perform a variable change. The new relaxed formulation possesses several favourable properties that allow us to easily compute a feasible solution which can be interpreted as a linear combination of the solutions in the RS. Then, we propose an algorithm to extract from such solution the RS by iteratively computing a defender's feasible solution with a suitable LP formulation.

Suppose that the defender's problem has some elements that can be interdicted (variables fixed to 0) and some others that cannot. We partition such elements into subsets I and J respectively. The attacker has a budget $B \in \mathbb{R}^+$ available to interdict the elements in set I by paying a cost $g_i \in \mathbb{Z}^+$. Notice that we are imposing that the vector g is integral because it will be essential for the correctness of the method.

Assume that the defender's problem can be modelled with the following formulation.

$$\text{opt} \quad \sum_{i\in I} c_i\, y_i + \sum_{j\in J} c_j\, y_j \tag{47.1}$$

s.t.

$$\sum_{i\in I} a_{ki}\, y_i + \sum_{j\in J} a_{kj}\, y_j = b_k \qquad \forall k \in K \tag{47.2}$$

$$y_i \in \{0,1\} \qquad \text{or } 0 \le y_i \le 1 \qquad \forall i \in I \tag{47.3}$$

$$y_j \in \mathbb{Z}^+ \qquad \text{or } y_j \ge 0 \qquad \forall j \in J \tag{47.4}$$

The variables y_i are the ones that the attacker can fix to 0 by paying interdiction cost g_i; variables y_j are other variables necessary to correctly model the problem. The formulation is a generic formulation with linear objective function and linear constraints (without loss of generality we consider only equality constraints). The variable domains are binary for y_i with $i \in I$ and non-negative integers for y_j with $j \in J$. However, since the matrix is totally unimodular, we consider its continuous relaxation.

In this environment, we denote the (*a priori* unknown) cardinality of Y' as $|Y'| = Q$ and assign an integer index $r \in R = \{1,2,\dots,Q\}$ to each solution $y^r \in Y'$. This way, formulation (31) slightly changes into

$$\begin{aligned}
&\min \sum_{i\in I} g_i \cdot x_i && && (48.1)\\
&\text{s.t.} && && \\
&\sum_{\substack{i\in I:\\ y_i^r=1}} x_i \geq 1 && \forall r \in R && (48.2)\\
&x_i \in \{0,1\} && \forall i \in I && (48.3)
\end{aligned}$$

where y_i^r is the value of variable y_i in solution y^r. Similarly, formulation (45) becomes

$$\begin{aligned}
&\max \sum_{r\in R} f_r && && (49.1)\\
&\text{s.t.} && && \\
&\sum_{\substack{r\in R:\\ y_i^r=1}} f_r \leq g_i && \forall i \in I && (49.2)\\
&f_r \geq 0 && \forall r \in R && (49.3)
\end{aligned}$$

Along the same lines as what we did in the previous section, we aim to determine the solutions to introduce in Y' and their dual values (f in formulation (49)) allowing us to exploit Theorem 5 also for this situation.

The following (non-linear) formulation describes the problem of finding Q solutions of the defender's problem and their dual values. Recall that Q is not known a priori, but for now, we assume its value and use it as a parameter of the formulation.

$$\begin{aligned}
\text{opt} \quad & \sum_{r\in R} f_r \left(\sum_{i\in I} c_i\, y_i^r + \sum_{j\in J} c_j\, y_j^r \right) && && (50.1)\\
\text{s.t.} \quad & && && \\
& \sum_{r\in R} f_r\, y_i^r \leq g_i && \forall i \in I && (50.2)\\
& \sum_{r\in R} f_r > B \left(\equiv \sum_{r\in R} f_r = \lfloor B+1 \rfloor =: B^+ \right) && && (50.3)\\
& \sum_{i\in I} a_{ki}\, y_i^r + \sum_{j\in J} a_{kj}\, y_j^r = b_k && \forall k \in K \quad \forall r \in R && (50.4)\\
& y_i^r \in \{0,1\} && \forall i \in I \quad \forall r \in R && (50.5)\\
& y_j^r \in \mathbb{Z}^+ && \forall k \in K \quad \forall r \in R && (50.6)\\
& f_r \geq 0 && \forall r \in R && (50.7)
\end{aligned}$$

Although this is a search problem (we aim to find any resilient sample), we model it as an optimization problem by introducing the objective function (50.1). While this function is not strictly necessary, it proves convenient for reasons that we will discuss later. The first constraints (50.2) state that for each interdictable element $i \in I$, the sum of the f_r variables such that $y_i^r = 1$ has to be smaller than or equal to the interdiction cost g_i. Constraint (50.3) imposes that the sum of the f_r variables must be bigger than B. Since strict inequalities are difficult to handle we just choose a value $B^+ > B$ and consider the equality version of the constraint. In particular, we define $B^+ := \lfloor B+1 \rfloor$ because we are interested in the smallest integer greater than B, and the reasons for this will be clearer later. At last, constraints (50.4) simply impose the original generic constraints (47.2) over all solutions y^r.

This is a troubling formulation for two main reasons: it is non-linear and requires to know the cardinality Q of Y'. But we can relax this formulation by applying a surrogate relaxation of the constraints (50.4), using the variables f_r as multipliers and summing over the index set R:

$$\begin{aligned}
\forall k \in K: \quad & \sum_{r\in R} f_r \left(\sum_{i\in I} a_{ki}\, y_i^r + \sum_{j\in J} a_{kj}\, y_j^r \right) = \sum_{r\in R} f_r\, b_k && \iff \\
& \sum_{r\in R}\sum_{i\in I} a_{ki}\, f_r\, y_i^r + \sum_{r\in R}\sum_{j\in J} a_{kj}\, f_r\, y_j^r = b_k \sum_{r\in R} f_r && \iff \\
& \sum_{i\in I} a_{ki} \sum_{r\in R} f_r\, y_i^r + \sum_{j\in J} a_{kj} \sum_{r\in R} f_r\, y_j^r = b_k\, B^+
\end{aligned}$$

Now, let's introduce the variables $\phi_i\ \forall i \in I$ and $\phi_j\ \forall j \in J$ defined as

$$\begin{aligned}
\phi_i &:= \sum_{r\in R} f_r\, y_i^r \le \sum_{r\in R} f_r = B^+ && \forall i \in I \\
\phi_j &:= \sum_{r\in R} f_r\, y_j^r && \forall j \in J
\end{aligned}$$

so that we can rewrite the surrogate generic constraint as

$$\sum_{i\in I} a_{ki}\, \phi_i + \sum_{j\in J} a_{kj}\, \phi_j = B^+\, b_k \qquad \forall k \in K$$

and, removing constraint (50.3) (further relaxing the formulation) we obtain

$$\begin{aligned}
\text{opt} \quad & \sum_{i\in I} c_i\, \phi_i + \sum_{j\in J} c_j\, \phi_j && && (51.1)\\
\text{s.t.} \quad & && && \\
& \phi_i \le g_i && \forall i \in I && (51.2)\\
& \sum_{i\in I} a_{ki}\, \phi_i + \sum_{j\in J} a_{kj}\, \phi_j = B^+\, b_k && \forall k \in K && (51.3)\\
& 0 \le \phi_i \le B^+ && \forall i \in I && (51.4)\\
& \phi_j \ge 0 && \forall j \in J && (51.5)
\end{aligned}$$

Notice that, formulation (51) is very similar to formulation (47), except for these reasons:

- the presence of constraints (51.2)
- the right-hand-side of every constraint (51.3) is the right-hand-side of (47.2) multiplied by B^+
- variables y_i are substituted with ϕ_i and y_j with ϕ_j
- the domain of the variables with index $i \in I$ is no longer $0 \le y_i \le 1$ but $0 \le \phi_i \le B^+$

Formulation (50) has the following useful property.

Proposition 14. *If B is insufficient to interdict all defender's solutions, then a feasible solution of formulation* (50) *exists, implying that also a solution of formulation* (51) *exists.*

Proof. Since B is insufficient to interdict the whole set Y, it means that Y is a resilient set itself. Now, consider the optimal dual variables f^* obtained solving formulation (49) using $Y' = Y$. We can show that if we use f^* as the variables f in formulation (50) and set $Y' = Y$, we obtain a feasible solution for the said formulation. Since f^* is a feasible solution, the inequality (49.2) holds and, therefore, constraints (50.2) are satisfied. Moreover, since f^* maximises the sum of its entries, also constraint (50.3) is also satisfied (recall that B is insufficient to interdict Y in its entirety). Trivially, since all solutions in Y are feasible, they fullfill constraints (47.2) and, therefore, constraints (50.4) are also satisfied. Given that, formulation (51) is obtained from (50) by relaxing some constraints and substituting the variables, we can produce a feasible solution also for formulation (51) simply from the definition of variables ϕ. □

Moreover, since formulations (47) and (51) are very similar, also this second proposition hold.

Proposition 15. *The coefficients matrix of formulation* (51) *is totally unimodular.*

Proof. Recall that the coefficients matrix of formulation (47) is totally unimodular. Formulation (51) shares the same structure except for constraints (51.2) but such constraints do not affect the total unimodularity. This is because they are just variable bounds and therefore they have a single 1 coefficient in every row of the matrix. Therefore, following the definition of totally unimodular matrix and the Laplace expansion of the determinant it is possible to prove that unimodularity is preserved. □

Therefore, if B^+ and all interdiction costs g are integer, then formulation (51) has the integrality property.

Solving formulation (51), we obtain the solution ϕ^* which "contains" the linear combination of all solutions in Y'. To "extract" such solutions from ϕ^* we slightly modify formulation (47) as follows.

$$\text{opt} \quad \sum_{i\in I} c_i\, y_i + \sum_{j\in J} c_j\, y_j \tag{52.1}$$

s.t.

$$\sum_{i\in I} a_{ki}\, y_i + \sum_{j\in J} a_{kj}\, y_j = b_k \qquad \forall k \in K \tag{52.2}$$

$$0 \le y_i \le \min\{1, \phi_i^*\} \qquad \forall i \in I \tag{52.3}$$

$$0 \le y_j \le \phi_j^* \qquad \forall j \in J \tag{52.4}$$

Once again, the coefficients matrix of this formulation is totally unimodular because we just introduced variable bounds and, since ϕ^* is integer, formulation (52) has integral solutions. Moreover, the following proposition, about the feasibility of the said formulation, holds.

Proposition 16. *If ϕ^* is a feasible solution for formulation* (51) *with respect to B^+, then formulation* (52) *admits a feasible solution.*

Proof. Consider the solution $y^* = \phi^*/B^+$. Since ϕ^* satisfies constraints (51.3) we know that

$$\begin{aligned}
\forall k \in K: \quad & \sum_{i\in I} a_{ki}\, \phi_i^* + \sum_{j\in J} a_{kj}\, \phi_j^* = B^+\, b_k & \iff \\
\iff & \sum_{i\in I} a_{ki}\, \phi_i^*/B^+ + \sum_{j\in J} a_{kj}\, \phi_j^*/B^+ = b_k & \iff \\
\iff & \sum_{i\in I} a_{ki}\, y_i^* + \sum_{j\in J} a_{kj}\, y_j^* = b_k &
\end{aligned}$$

Therefore, y^* satisfies constraints (52.2). Moreover, since $\phi^* \ge 0$ and $B^+ \ge 1$ it follows that $0 \le y^* = \phi^*/B^+ \le \phi^*$. Moreover, for each $i \in I$, $\phi_i^* \le B^+$ and therefore $y_i^* = \phi_i^*/B^+ \le 1$ satisfying all variable bounds. Therefore y^* is a feasible (possibly fractional) solution of formulation (52). □

Given that, for any positive integer B^+, the formulation (52) admits a feasible solution and has the integrality property, we can extract the first solution from ϕ^* by computing the optimal solution of (52). Then we update ϕ^* by removing the solution from the linear combination and repeat the process until all solutions are extracted. Algorithm 13 describes the procedure.

```
Algorithm Resilient Sample for TUBIV($B \in \mathbb{R}^+, g : I \to \mathbb{Z}^+$):
    $B^+ := \lfloor B + 1 \rfloor$
    $Y' = \emptyset$
    $\tilde{f}_r = 0 \quad \forall r \in \mathbb{Z}^+$ // Implicitly
    Solve (51) using $B^+$ and $g$ as parameters, producing $\phi^*$
    $r = 0$
    $\phi^r = \phi^*$
    $B_r = B^+$
    while $B_r > 0$ do
        $r = r + 1$
        Solve (52) using $\phi^{r-1}$ as parameters, producing $y^r$
        $\delta_I = \min_{i \in I : y_i^r = 1} \phi_i^{r-1}$
        $\delta_J = \min_{j \in J : y_j^r \geq 1} \phi_i^{r-1} / y_j^r$
        $\tilde{f}_r = \min \{\delta_I, \lfloor \delta_J \rfloor\}$
        $Y' = Y' \cup \{y^r\}$
        for $i \in I$ do $\phi_i^r = \phi_i^{r-1} - \tilde{f}_r\, y_i^r$
        for $j \in J$ do $\phi_j^r = \phi_j^{r-1} - \tilde{f}_r\, y_j^r$
        $B_r = B_{r-1} - \tilde{f}_r$
    end
    return $Y', \tilde{f}$
```

Algorithm 13: Algorithm for computing a resilient sample for any TUBIV problem.

Initially, Y' is set to $\emptyset$ as well as $\tilde{f}_r = 0$ for any index $r \in \mathbb{Z}^+$ since we do not know Q a priori (obviously we do not do it explicitly). Then, we solve formulation (51) using B^+ to obtain ϕ^*. The index r is initialised to 0 and corresponds to the number of the current iteration of the main cycle and the index of the current solution extracted from ϕ^*. We set the parameters ϕ^r and B_r to the initial values ϕ^* and B^+ respectively. Then, the while loop begins. First, we increase r and solve formulation (52) using ϕ^{r-1} to obtain y^r. Then, we compute δ_I which is the minimum value of ϕ_i^{r-1} among all $i \in I$ for which $y_i^r = 1$ and, similarly, we compute δ_J which is the minimum value of ϕ_j^{r-1}/y_j^r among all $j \in J$ for which $y_j^r \geq 1$ (recall that y_j^r might be strictly bigger than 1). The dual value $\tilde{f}_r$ is computed as the minimum between δ_I (which is already integer) and the floor of δ_J. Then, we insert y^r in Y'. At this point, we update ϕ^r as follows: for each $i \in I$, we decrease ϕ_i^{r-1} by $\tilde{f}_r\, y_i^r$; for each $j \in J$, we decrease ϕ_j^{r-1} by $\tilde{f}_r\, y_j^r$. At last, we set the value B_r to the value of the previous iteration B_{r-1} decreased by $\tilde{f}_r$. These updates of B_r and ϕ^r allow us to proceed with the computation because they guarantee that ϕ^r, after the update, is a feasible (integer) solution of (51) using B_r as parameter (see Proposition 17). Finally, when $B_r \leq 0$, it means that the sum of all $\tilde{f}_r$ produced so far equals B^+ and therefore we can return Y' alongside with the dual values.

The correctness of the algorithm is guaranteed by the following propositions.

Proposition 17. *At each iteration of Algorithm 13, ϕ^r is a feasible solution of* (51) *using B_r as parameter.*

Proof. By induction on r we can see that when $r = 0$, the hypothesis is trivially true. When $r > 0$, by inductive hypothesis, ϕ^{r-1} is a solution of (51) with parameter B_{r-1} and therefore constraints (51.3) are satisfied.

$$\forall k \in K: \quad \sum_{i \in I} a_{ki}\, \phi_i^{r-1} + \sum_{j \in J} a_{kj}\, \phi_j^{r-1} = B_{r-1}\, b_k$$

Now, we consider the left-hand-side of the same constraints for ϕ^r:

$$\begin{aligned}
\forall k \in K: \quad & \sum_{i \in I} a_{ki}\, \phi_i^r + \sum_{j \in J} a_{kj}\, \phi_j^r && = \\
&= \sum_{i \in I} a_{ki}\, (\phi_i^{r-1} - \tilde{f}_r\, y_i^r) + \sum_{j \in J} a_{kj}\, (\phi_j^{r-1} - \tilde{f}_r\, y_j^r) && = \\
&= \sum_{i \in I} a_{ki}\, \phi_i^{r-1} - \sum_{i \in I} a_{ki}\, \tilde{f}_r\, y_i^r + \sum_{j \in J} a_{kj}\, \phi_j^{r-1} - \sum_{j \in J} a_{kj}\, \tilde{f}_r\, y_j^r && = \\
&= B_{r-1}\, b_k - \tilde{f}_r \left(\sum_{i \in I} a_{ki}\, y_i^r + \sum_{j \in J} a_{kj}\, y_j^r \right) && = \\
&= B_{r-1}\, b_k - \tilde{f}_r\, b_k = b_k\, (B_{r-1} - \tilde{f}_r) = B_r\, b_k
\end{aligned}$$

This, proves that ϕ^r satisfies constraints (51.3). Moreover, since $\phi^r \le \phi^{r-1} \le g$ we know that also constraints (49.2) are satisfied. The only thing left to prove is that $\phi^r \ge 0$: we know that $\phi^r = \phi^{r-1} - \tilde{f}_r\, y^r$ therefore we have to check that $\tilde{f}_r\, y^r \le \phi^{r-1}$. Recall that $\tilde{f}_r \le \phi_i^{r-1}$ for all $i \in I : y_i^r = 1$ and $\tilde{f}_r \le \phi_j^{r-1}\, y_j^r$ for all $j \in J : y_j^r \ge 1$. Therefore, $\tilde{f}_r\, y_i^r \le \phi_i^{r-1}$ and $\tilde{f}_r\, y_j^r \le \phi_j^{r-1}$. □

Proposition 18. *At each iteration of Algorithm 13, all values of $\phi^r, y^r, \tilde{f}_r$ and B_r are positive integers.*

Proof. By induction on r we can check that when $r = 0$ the values $\phi^r = \phi^*$ and $B_r = B^+$ are integers because of Proposition 15, given that B^+ is chosen integer ($B^+ = \lfloor B + 1 \rfloor$). If $r > 0$, the first thing that is computed is y^r: we know from Proposition 16 (whose conditions are met due to the previous point) that a feasible solution y^r exists and that formulation (52) has integrality property. Moreover, we can assume that $y^r \ne 0$ since, otherwise, $\{y^r\}$ would be a resilient sample alone. Since $0 \ne y^r \le \phi^{r-1}$, we know that $\tilde{f}_r > 0$. Moreover, $\tilde{f}_r$ is integral by construction. As we can see, ϕ^r is obtained from subtractions and multiplications of integers, therefore is integral itself and never smaller than 0 (proven in the previous point). At last, B_r is integer (because $\tilde{f}_r$ is integer) and greater than 0 because of the termination condition of the while cycle. □

Proposition 19. *At the end of Algorithm 13, the sum of all $\tilde{f}_r$ computed so far is equal to B^+.*

Proof. At the end of the algorithm, we exit the cycle with r reaching the final value Q, because $B_Q \leq 0$. First, we prove that $B_Q = 0$ and cannot be negative. This is because for any $r > 0, \exists i \in I: \tilde{f}_r \leq \phi_i^{r-1} \leq B_{r-1}$ since ϕ^r is a feasible solution of formulation (51) with parameter B_{r-1}. Therefore, $B_r = B_{r-1} - \tilde{f}_r \geq 0$ for all $r > 0$ implying that $B_Q = 0$. Knowing that $\tilde{f}_r = B_{r-1} - B_r$ for any $r > 0$, we can conclude that

$$\sum_{r\in R} \tilde{f}_r = \sum_{r\in R} (B_{r-1} - B_r) = B_0 - B_Q = B^+$$

□

The solutions obtained can be recomposed again as described in the proposition below.

Proposition 20. *The linear combination of the solutions in Y' using $\tilde{f}$ as coefficients is a feasible solution of formulation* (51)*, using parameter B^+, equivalent to ϕ^*.*

Proof. Recall that in Algorithm 13, the relationship between ϕ^r and ϕ^{r-1}, for any $r > 0$, is

$$\phi^r = \phi^{r-1} - \tilde{f}_r\, y^r \iff \tilde{f}_r\, y^r = \phi^{r-1} - \phi^r$$

Now we can show that

$$\sum_{r\in R} \tilde{f}_r\, y^r = \sum_{r\in R} (\phi^{r-1} - \phi^r) = \phi^0 - \phi^Q = \phi^* - \phi^Q$$

From Proposition 17 we know that ϕ^Q is a solution of formulation (51) with respect to $B_Q = 0$. Therefore, constraints (51.3) for $\phi^* - \phi^Q$ is

$$\begin{aligned}
\forall k \in K: \quad & \sum_{i\in I} a_{ki}(\phi_i^* - \phi_i^Q) + \sum_{j\in J} a_{ki}(\phi_j^* - \phi_j^Q) & = \\
= & \sum_{i\in I} a_{ki}\,\phi_i^* - \sum_{i\in I} a_{ki}\,\phi_i^Q + \sum_{j\in J} a_{ki}\phi_j^* - \sum_{j\in J} a_{ki}\phi_j^Q & = \\
= & B^+\, b_k - B_Q\, b_k = B^+\, b_k &
\end{aligned}$$

Since $B_Q = 0$ we also know that $\forall i \in I: \ 0 \leq \phi_i^Q \leq B_Q = 0 \implies \phi_i^Q = 0$. So, $\forall i \in I: \ \phi_i^* - \phi_i^Q = \phi_i^* \leq g_i$, satisfying also constraints (49.2) and the variable bounds. □

Finally, we can prove that the set Y' returned by Algorithm 13 is a resilient sample with respect to B.

Theorem 6. *Given a TUBIV problem with a budget $B \in \mathbb{R}^+$, the set Y' returned by Algorithm 13 is a resilient sample.*

Proof. Similarly to Theorem 5, we want to prove that

$$B < \sum_{r \in R} \tilde{f}_r \leq (49)^* \leq g(x^*)$$

where $(49)^*, g(x^*)$ are the optimal values of formulations (49) and (48) respectively.

We know from Proposition 19 that

$$\sum_{r \in R} \tilde{f}_r = B^+ > B$$

Moreover, we know from Proposition 20 that

$$\forall i \in I: \sum_{\substack{r \in R: \\ y_i^r = 1}} \tilde{f}_r = \sum_{r \in R} \tilde{f}_r y_i^r = \phi_i^* - \phi_i^Q \leq g_i$$

proving that $\tilde{f}$ is a feasible solution of formulation (49). At last, we know that the optimum of formulation (49) is smaller than or equal to the optimum of formulation (48) because the former is the dual of the continuous relaxation of the latter. That said, since the minimum budget to interdict Y' is strictly bigger than the given budget B, we know that Y' is a resilient sample. □

7.3.1 SPIP as a special case

Now we show that the technique showed in section 7.2 is simply a special case of the one in this section.

Consider the classic LP formulation for the shortest path problem

$$\min \sum_{(i,j) \in A} c_{ij} y_{ij}$$

s.t.

$$\sum_{i \in \delta_k^-} y_{ik} - \sum_{j \in \delta_k^+} y_{kj} = \begin{cases} -1, & k = s \\ 1, & k = t \\ 0, & \text{otherwise} \end{cases} \qquad k \in N$$

$$y_{ij} \in \{0,1\} \text{ or } 0 \leq y_{ij} \leq 1 \qquad (i,j) \in A$$

By applying the new technique we obtain

$$\begin{aligned}
&\min \sum_{(i,j)\in A} c_{ij}\phi_{ij} \\
&\text{s.t.} \\
&\quad \sum_{i\in\delta_k^-}\phi_{ik} - \sum_{j\in\delta_k^+}\phi_{kj} = \begin{cases} -B^+, & k=s \\ B^+, & k=t \\ 0, & \text{otherwise} \end{cases} && k\in N \\
&\quad 0 \le \phi_{ij} \le \min\{B^+, g_{ij}\} && (i,j)\in A
\end{aligned}$$

which is a $s-t$ min-cost flow formulation with flow value B^+. This adds to the various pieces of evidence supporting the relationship between the min-cost flow and the SPIP like the one in Fulkerson and Harding (1977) and the lagrangean relaxation in Section 5.3.1.

7.3.2 A counterexample beyond the procedure's applicability

Unfortunately, the technique presented in this section strongly relies on the problem to be TUBIV. In fact, we can show an example where we consider an ILP defender's problem (without a totally unimodular matrix) for which we cannot exploit the last technique.

Consider the Set Cover instance

$$\begin{aligned}
\min\quad & y_1+y_2+y_3 \\
\text{s.t.}\quad & \\
& y_1+y_2-s_1=1 \\
& y_1+y_3-s_2=1 \\
& y_2+y_3-s_3=1 \\
& y_1,y_2,y_3\in\{0,1\} \\
& s_1,s_2,s_3\ge 0
\end{aligned}$$

and the Interdiction problem where $B=g_1=g_2=g_3=1$. So we choose $B^+=2>B$ obtaining

$$\begin{aligned}
\min\quad & \phi_1+\phi_2+\phi_3 \\
\text{s.t.}\quad & \\
& \phi_1+\phi_2-\sigma_1=2 \\
& \phi_1+\phi_3-\sigma_2=2 \\
& \phi_2+\phi_3-\sigma_3=2 \\
& \phi_1\le 1 \\
& \phi_2\le 1 \\
& \phi_3\le 1 \\
& \phi_1,\phi_2,\phi_3,\sigma_1,\sigma_2,\sigma_3\ge 0
\end{aligned}$$

whose optimal solution is $\phi_1 = \phi_2 = \phi_3 = 1$ and $\sigma_1 = \sigma_2 = \sigma_3 = 0$. Since σs are all zero, it means that the s variables have to be zero in formulation (52), obtaining a problem of the form

$$\begin{aligned} y_1 + y_2 &= 1 \\ y_2 + y_3 &= 1 \\ y_1 + y_3 &= 1 \end{aligned}$$

which is not feasible when y are binary, showing that there are instances of problems for which our procedure fails to produce a RS.

7.4 Resilient Sample for any Binary Interdiction Problem

As showed in Section 7.3.2, the previous technique is not able to find a RS for any binary interdiction problem. Of course, one could still extract a number of feasible solutions with Algorithm 13 and stop whenever formulation (52) fails to produce a feasible solution. If so, even if the produced set is not a RS, we can use it as base to produce one.

In this section we will use a classical constraint-separation approach to generate a resilient sample for a generic binary interdiction problem. This approach is still *heuristic*, meaning that it does not provide any guarantee that the produced set is a RS. The approach involves considering the continuous relaxation of the covering decomposition and utilizing it as a Master problem. After obtaining an optimal solution for the Master, we then address a Separation problem, which entails computing a feasible solution for the defender that corresponds to the most violated covering constraint associated with the Master's solution, and subsequently adding this constraint to the Master. In a certain way, this approach is similar to Algorithm 9 (from Chapter 6) with the main difference that here we consider a continuous relaxation of the covering decomposition instead of the computation of the RRR.

Similarly to section 7.3, we provide a generic MILP formulation for the defender's problem. Once again, the elements in the set I are the ones that can be interdicted whereas the ones in J cannot.

$$\text{opt} \sum_{i \in I} c_i y_i + \sum_{j \in J} c_j y_j \tag{55.1}$$

s.t.

$$\sum_{i \in I} a_{ki} y_i + \sum_{j \in J} a_{kj} y_j = b_k \qquad \forall k \in K \tag{55.2}$$

$$y_i \in \mathbb{Z}^+ \text{ or } y_i \in \mathbb{R}^+ \qquad \forall i \in I \tag{55.3}$$

$$y_j \in \mathbb{Z}^+ \text{ or } y_j \in \mathbb{R}^+ \qquad \forall j \in J \tag{55.4}$$

Furthermore, we redefine the formulations (31) and (48) for the most general case.

$$\min \sum_{i \in I} g_i x_i \tag{56.1}$$

s.t.

$$\sum_{\substack{i \in I: \\ y_i > 0}} x_i \geq 1 \qquad y \in Y' \tag{56.2}$$

$$x_i \in \{0,1\} \qquad i \in I \tag{56.3}$$

And its continuous relaxation.

$$\min \sum_{i \in I} g_i x_i \tag{57.1}$$

s.t.

$$\sum_{\substack{i \in I: \\ y_i > 0}} x_i \geq 1 \qquad y \in Y' \tag{57.2}$$

$$x_i \geq 0 \qquad i \in I \tag{57.3}$$

We aim to populate the set Y' iteratively by solving the following separation problem

$$\min \sum_{i \in I} \tilde{x}_i \gamma_i \tag{58.1}$$

s.t.

$$\sum_{i \in I} a_{ki} y_i + \sum_{j \in J} a_{kj} y_j = b_k \qquad \forall k \in K \tag{58.2}$$

$$y_i \leq \bar{y}_i \gamma_i \qquad \forall i \in I \tag{58.3}$$

$$y_i \in \mathbb{Z}^+ \text{ or } y_i \in \mathbb{R}^+ \qquad \forall i \in I \tag{58.4}$$

$$y_j \in \mathbb{Z}^+ \text{ or } y_j \in \mathbb{R}^+ \qquad \forall j \in J \tag{58.5}$$

$$\gamma_i \in \{0,1\} \qquad \forall i \in I \tag{58.6}$$

where $\tilde{x}$ is the optimum of formulation (57). To correctly model the objective function, we introduce the binary variables γ_i $\forall i \in I$ which assume value 1 whenever $y_i > 0$ due to constraints (58.3) and assume value 0 otherwise due to the minimisation and the non-negativity of $\tilde{x}$. By finding the optimal solution of formulation (58) we identify the most violated constraint (57.2) with respect to $\tilde{x}$.

From these considerations we derive Algorithm 14. At first, the optimal defender's solution y^* is computed by solving formulation (55). The set of candidate solutions T is initialised to $\{y^*\}$ and then we enter the main loop. Inside the loop we compute the optimal solution $\tilde{x}$ of formulation (57) only considering the covering constraints associated with the solutions in T. If the minimum budget to interdict T with the relaxed attack $\tilde{x}$ is greater than B we know that T contains a resilient sample and, therefore, we exit the loop. Otherwise, we solve the pricing problem by computing the optimal solution $\tilde{y}$ of formulation (58) using $\tilde{x}$ as shadow prices. If the sum of the $\tilde{x}_i$ such that $\tilde{y}_i > 0$ is greater than 1, it means that the "most violated constraint" is not

```
1  Algorithm Resilient Sample for Binary Interdiction($B \in \mathbb{R}^+$, $g : I \to \mathbb{R}^+$):
2      Solve formulation (55), obtaining $y^*$
3      $T = \{y^*\}$
4      loop:
5          Solve formulation (57) setting $Y' = T$, obtaining $\tilde{x}$
6          if $\sum_{i \in I} g_i \tilde{x}_i > B$ then
7              return $Y' = \{y \in T : \sum_{i \in I: y_i > 0} \tilde{x}_i = 1\}$
8          end
9          Solve formulation (58) using $\tilde{x}$, obtaining $\tilde{y}$
10         if $\sum_{i \in I: \tilde{y}_i > 0} \tilde{x}_i \geq 1$ then
11             exit loop
12         end
13         $T = T \cup \{\tilde{y}\}$
14         Heuristically generate solutions $\hat{y} \in Y$ such that $\sum_{i \in I: \hat{y}_i > 0} \tilde{x}_i < 1$
15         Insert such solutions into $T$
16     end
17     return $T$ as heuristic set
```

Algorithm 14: Algorithm for computing a resilient sample for the a generic Binary Interdiction problem

violated, implying that it is not possible to increase the objective of formulation (57). In such case, we return T as a heuristic set. Otherwise, we insert $\tilde{y}$ in T and also consider other heuristic solutions whose associated covering constraint is violated (maybe exploiting the diversification mechanisms in Section 6.4). Once we exit the loop, because the objective of $\tilde{x}$ is greater than B, we select all solutions in T associated with an active covering constraint with respect to $\tilde{x}$. This set is a RS because to interdict all of its solutions we need more budget than B.

Considering the unfortunate case where we cannot find any solution which violates constraint (57.2), it is unknown a priori if T is a RS. In fact, since (57) is a relaxation of (56), the optimal solution of the latter using $Y' = T$ might be $> B$ producing a RS. Moreover, if T is not a RS, one could initialise $Y' = T$ in Algorithm 13 and try to compute a RS exploiting the exact approach presented in Chapter 6.

7.5 Resilient Sample for Binary Fortification Problems

Since Fortification Problems can be interpreted as interdiction problems where the defender solves another Interdiction Problem, we can extend this work also for the fortification step.

Once again, we are interested in a RS but, this time, instead of sampling the set Y of defender's solutions, we will sample the set X_B of feasible attacks looking for some $X' \subset X_B$. X_B has a very simple knapsack structure because the solutions are subsets of the interdictable elements whose interdiction cost does not exceed the attack budget B.

Similarly to the case of the Shortest Path Interdiction Problem, if the fortification budget P is big enough to immunise some optimal solution (of the last level problem) we could simply fortify every element in that solution and the problem would be trivial.

Therefore, we are interested in some optimal solution $y^* \in Y$ of minimum fortification cost (the cost you have to pay in order to fortify the entire solution). Formally

$$\begin{aligned} c^* &:= \min_{y \in Y} c(y) \\ Y^* &:= \{y \in Y \mid c(y) = c^*\} \\ y^* &:= \arg\min_{y \in Y^*} \sum_{i \in y} p_i \end{aligned}$$

In the specific case of the Shortest Path Fortification Problem, finding the shortest path which minimises also the sum of the fortification costs can be done easily by modifying Dijkstra's algorithm so that comparisons will be hierarchical (path cost is more important that fortification cost) instead of scalar.

A very trivial initial sample can be computed as follows

```
1 Algorithm FortificationSet($G = (N, A)$, $f : A \to \mathbb{R}^+$, $B_F \in \mathbb{R}^+$):
2     $y^* = \arg\min_{y \in Y^*} \sum_{i \in y} f_i$
3     if $\sum_{i \in y^*} f_i \le B_F$ then return trivial instance
4     $X' = \{\{i\} \mid i \in y^*\}$
5     return $X'$
```

Algorithm 15: Algorithm for the computation of RS X'

Algorithm 15 is pretty simple: line 2 is the computation of an optimal solution of minimum fortification cost (modified Dijkstra's algorithm for the Shortest Path Fortification Problem), then we check for the triviality of the instance and at last we return X' as the set of singletons of the elements in y^*.

The set X' is indeed a RS because, trivially, if the defender tries to interdict any of the attacks $\{i\} = x \in X'$, he must fortify i. This means that, in order to interdict every attack $x \in X'$ we need to fortify all the elements in the optimal solution y^* which is impossible since $\sum_{i \in y^*} p_i > P$.

There are some improvements that can be considered:

- if P is much smaller than $\sum_{i \in y^*} p_i$ we can try to erase some singletons from X' in order to decrease its size but maintaining the property that $\sum_{\{i\} \in X'} p_i > P$
- we can populate the singletons in X' with other elements maintaining two properties:
    1. $\forall x \in X' : \sum_{i \in x} p_i \leq P$ (feasible attacks)
    2. $\forall x_1, x_2 \in X' : x_1 \cap x_2 = \emptyset$ (independent attacks)

The reason for these two improvements is that this way we can obtain stronger attacks which lead to a stronger super-optimal bound. In fact the lower bound (because we are minimising) produced by X' is the objective of the attack $x \in X'$ that produces the smallest defender's solution

$$\min_{x \in X'} \min_{y \in Y(x)} c^\top y \leq \min_{w \in W_P} \max_{x \in X_B(w)} \min_{y \in Y(x)} c^\top y = c(w^*)$$

This is true because since X' cannot be completely interdicted by the defender, we know that at least one of the attacks will remain admissible in the optimal solution. The one with smallest cost is a valid lower bound.

The Algorithm 16 is an improvement of Algorithm 15. The first lines of Algorithm 16 are Algorithm 15 but instead of X' we obtain $\hat{X}$. Then the first operations (lines 5-10) are to remove from $\hat{X}$ all the attacks $\tilde{x}$ such that $\hat{X}$ remains a resilient set (line 8), starting from the attacks that produce the solution of minimum cost. Then we initialise $X' := \emptyset$ which is the set of the attacks that will be returned at the end and it is a set of attacks that do not change anymore during the execution of the algorithm. We compute also the set L of all the elements outside the attacks in both X' and $\hat{X}$, they will be the elements candidate for being inserted in the attacks of $\hat{X}$. Then, in the while loop (lines 13-29), we extract the attack $\tilde{x}$ that produces the minimum cost solution $\tilde{y}$ among all attacks in $\hat{X}$. In the for loop (lines 17-24) we try to insert in $\tilde{x}$ some element of $\tilde{y}$ which is not in any other attack, preserving its feasibility (line 18). If none of the candidate elements can be feasibly inserted, the attack $\tilde{x}$ is removed from $\hat{X}$ and inserted in X'.

The algorithm returns a resilient set of disjoint (none of them have any element in common) feasible attacks X' of good quality (heuristically). Moreover the cardinality of X' is tractable:

$$|X'| \leq |y^*| \leq \max_{y \in Y} |y|$$

The fact that the attacks are disjoint is crucial for the resilience of X' because otherwise it could be that, fortifying an element that occurs in more than one attack, X' might be completely interdicted.

The attacks could be furthermore improved by some exchange heuristics. Moreover, this approach can be exploited to compute a RS for any problem with a knapsack solution space.

```
Algorithm FortificationSetImproved($G = (N, A), c : A \to \mathbb{R}^+, f : A \to \mathbb{R}^+, B_I \in \mathbb{R}^+, B_F \in \mathbb{R}^+$):
    $y^* = \arg\min_{y \in Y^*} \sum_{i \in y} p_i$
    if $\sum_{i \in y^*} p_i \leq P$ then return $\textit{trivial instance}$
    $\hat{X} = \{\{i\} \mid i \in y^*\}$
    loop:
        $\tilde{x} = \arg\min_{x \in \hat{X}} \min_{y \in Y(x)} c(y)$
        $\tilde{y} = \arg\min_{y \in Y(x)} c(y)$
        if $\sum_{\{i\} \in \hat{X} \setminus \{\tilde{x}\}} p_i \leq P$ then exit loop
        $\hat{X} = \hat{X} \setminus \{\tilde{x}\}$
    end
    $X' = \emptyset$
    $L = A \setminus \bigcup_{x \in \hat{X}} x$
    while $\hat{X} \neq \emptyset$ do
        $\tilde{x} = \arg\min_{x \in \hat{X}} \min_{y \in Y(x)} c(y)$
        $\tilde{y} = \arg\min_{y \in Y(\tilde{x})} c(y)$
        change = false
        for $j \in \tilde{y} \cap L$ do
            if $\sum_{i \in \tilde{x}} p_i + p_j \leq P$ then
                $\tilde{x} = \tilde{x} \cup \{j\}$
                $L = L \setminus \{j\}$
                change = true
                break
            end
        end
        if change == false then
            $X' = X' \cup \{\tilde{x}\}$
            $\hat{X} = \hat{X} \setminus \{\tilde{x}\}$
        end
    end
    return $X'$
```

Algorithm 16: Algorithm for the computation of an improved resilient sample X'

Chapter 8

Conclusions

In this thesis, we described several optimisation problems related to network safety, focusing on the Weighted Safe Set Problem (WSSP), the Shortest Path Interdiction Problem (SPIP) and the Set Covering Interdiction Problem (SCIP).

In Chapter 2 we described the WSSP and surveyed the related literature. Then, we described several linear programming (continuous, integer, mixed-integer) formulations and introduced several benchmark instances, both from the literature and newly generated ones.

In Chapter 3 we proposed an exact combinatorial branch-and-bound algorithm to solve the WSSP. For this algorithm we devised two main versions: the first employs a strict lower bound procedure that strongly affects many other aspects of the branch-and-bound; the second applies a flexible lower bound procedure that allows more freedom in the other aspects of the branch-and-bound even if the bound is theoretically weaker. We tested the algorithm on the instances from the literature to show that both versions outperform the state-of-the-art methods. Then, we also introduced new harder instances and showed that the flexible version outperforms the strict one.

In Chapter 4 we presented several constructive-destructive metaheuristics for the WSSP as well as a Scatter Search heuristic. We tuned their hyper-parameters in order to find the best performing versions in terms of quality of the produced solution in a given time. Then, we compared the new heuristics with each other and with the only available heuristic from the literature. The results showed that the new heuristics outperform the old one and that, among them, the best performing one is the Scatter Search.

In Chapter 5 we shift the focus to Interdiction problems. In particular, we dwell on the hard interdiction problems where the attacker blocks/erases elements from the instance. We present a literature survey on these problems and other related problems like the fortification ones. Then, we delve into details for the SPIP and SCIP.

In Chapter 6 we described the state-of-the-art exact methods for solving interdiction problems. Then, we proposed an adaptation of these approaches for the specific case of hard interdiction, along with an original framework that can be interpreted as a generalization of one of the approaches from the literature. Moreover, we presented 3 diversification techniques to speed up the computation of the optimal solution. We showed the improvements of the new methods by implementing them for both the SPIP and the SCIP. For the SPIP, we ran the algorithms on grid instances from the literature showing that the new framework is competitive with the state-of-the-art. Moreover, we compared the 3 diversification techniques with a few others available from the literature for the SPIP. For the SCIP, we performed some experimental computation on new generated instances, yielding results where the original framework outperforms the other approaches.

In Chapter 7 we introduced the concept of Resilient Sample (RS) which is a subset of defender's feasible solutions such that no feasible attack is able to interdict all of them simultaneously. We provide a method for computing a RS for the SPIP and show that the same method can be applied to the maximum flow interdiction problem. Then we generalise the same method for a broader class of interdiction problems that we called TUBIV. Then we showed the limits of the proposed method and provided an alternative heuristic method which works for any interdiction problem. At last we described how to build a RS method for fortification problems and other problems with a knapsack solution space.

Acknowledgements

I would like to express my heartfelt gratitude to Professor Roberto Cordone for his expert guidance, which has significantly raised my understanding of scientific research. I am particularly thankful for his ability to nurture my ideas, transforming my youthful enthusiasm into solid scientific discoveries I can truly be proud of.

I also extend my thanks to Dr. Pierre Hosteins for introducing me to the field of interdiction problems and the safe set problem, and for including me in his projects.

I am grateful to Professor Austin Buchanan for our fruitful conversation, which inspired one of the formulations presented in this thesis.

A special thank you goes to my colleagues in OptLab: Cristina, Daniel, Marco, Michele, and Rosario. The wonderful experiences we shared during my doctoral years have been invaluable.

Lastly, I would like to thank AIRO and AIROYoung for their wonderful initiatives and conferences, where I had the opportunity to attend numerous interesting presentations that have partly inspired my research, and where I met many people with whom I shared precious moments.

Bibliography

Abdolahzadeh, A., Aman, M., Tayyebi, J., 2020. Minimum st-cut interdiction problem. Computers & Industrial Engineering 148, 106708.

Àgueda, R., Cohen, N., Fujita, S., Legay, S., Manoussakis, Y., Matsui, Y., Montero, L., Naserasr, R., Ono, H., Otachi, Y., Sakuma, T., Tuza, Z., Xu, R., 2018. Safe sets in graphs: Graph classes and structural parameters. Journal of Combinatorial Optimization 36, 1221–1242.

Ahuja, R.K., Magnanti, T.L., Orlin, J.B., 1993. Network Flows: Theory, Algorithms, and Applications. Prentice-Hall, Inc., USA.

Assimakopoulos, N., 1987. A network interdiction model for hospital infection control. Computers in Biology and Medicine 17, 413–422.

Ball, M.O., Golden, B.L., Vohra, R.V., 1989. Finding the most vital arcs in a network. Operations Research Letters 8, 73–76.

Bapat, R.B., Fujita, S., Legay, S., Manoussakis, Y., Matsui, Y., Sakuma, T., Tuza, Z., 2018. Safe sets, network majority on weighted trees. Networks 71, 81–92.

Bapat, R.B., Fujita, S., Legay, S., Manoussakis, Y., Y.Matsui, Sakuma, T., Tuza, Z., 2016. Network majority on tree topological network. Electronic Notes in Discrete Mathematics 54, 79–84.

Beasley, J., 1987. An algorithm for set covering problem. European Journal of Operational Research 31, 85–93.

Belmonte, R., Hanaka, T., Katsikarelis, I., Lampis, M., Ono, H., Otachi, Y., 2020. Parameterized complexity of safe set. Journal of Graph Algorithms and Applications 24, 215–245.

Boggio Tomasaz, A., Carvalho, M., Cordone, R., Hosteins, P., 2024. On the completeness of several fortification-interdiction games in the polynomial hierarchy. URL: `https://arxiv.org/abs/2406.01756`.

Boggio Tomasaz, A., Cordone, R., 2024a. A revisited branch and bound method for the weighted safe set problem, in: Proceedings of the 13th International Conference on Operations Research and Enterprise Systems. 1, SciTePress. pp. 113–122.

Boggio Tomasaz, A., Cordone, R., 2024b. A scatter search metaheuristic and improvements to an exact algorithm for the weighted safe set problem. Submitted to Communications in Computer and Information Science, Springer.

Boggio Tomasaz, A., Cordone, R., Hosteins, P., 2023a. A combinatorial branch and bound for the safe set problem. Networks 81, 445–464.

Boggio Tomasaz, A., Cordone, R., Hosteins, P., 2023b. Constructive–destructive heuristics for the safe set problem. Computers & Operations Research 159, 106311.

Brown, G., Carlyle, M., Salmerón, J., Wood, K., 2006. Defending critical infrastructure. Interfaces 36, 530–544.

Cappanera, P., Scaparra, M.P., 2011. Optimal allocation of protective resources in shortest-path networks. Transportation Science 45, 64–80.

Caprara, A., Carvalho, M., Lodi, A., Woeginger, G.J., 2013. A complexity and approximability study of the bilevel knapsack problem, in: Integer Programming and Combinatorial Optimization, pp. 98–109.

Caprara, A., Carvalho, M., Lodi, A., Woeginger, G.J., 2014. A study on the computational complexity of the bilevel knapsack problem. SIAM Journal of Optimization 24, 823–838.

Caprara, A., Carvalho, M., Lodi, A., Woeginger, G.J., 2016. Bilevel knapsack with interdiction constraints. INFORMS Journal on Computing 28, 319–333.

Church, R.L., Scaparra, M.P., Middleton, R.S., 2004. Identifying critical infrastructure: the median and covering facility interdiction problems. Annals of the Association of American Geographers 94, 491–502.

Cordone, R., Franchi, D., 2023. The safe set problem on particular graph classes, in: Proceedings of the 19th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW), Garmisch-Partenkirchen, Germany. pp. 1–4.

Della Croce, F., Scatamacchia, R., 2020. An exact approach for the bilevel knapsack problem with interdiction constraints and extensions. Mathematical Programming 183, 249–281.

Dunn, O.J., 1961. Multiple comparisons among means. Journal of the American Statistical Association 56, 52–64.

Ehard, S., Rautenbach, D., 2020. Approximating connected safe sets in weighted trees. Discrete Applied Mathematics 281, 216–223.

Ehrgott, M., 2005. Multicriteria Optimization. Springer-Verlag, Berlin, Heidelberg.

Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Fischetti, M., Ljubić, I., Monaci, M., Sinnl, M., 2019. Interdiction games and monotonicity, with application to knapsack problems. INFORMS Journal on Computing 31, 390–410.

Fröhlich, N., Ruzika, S., 2021. On the hardness of covering-interdiction problems. Theoretical Computer Science 871, 1–15.

Fujita, S., MacGillivray, G., Sakuma, T., 2014. Bordeaux graph workshop: Safe set problem on graphs. https://bgw.labri.fr/2014/bgw2014-booklet.pdf.

Fujita, S., MacGillivray, G., Sakuma, T., 2016. Safe set problem on graphs. Discrete Applied Mathematics 215, 106–111.

Fulkerson, D.R., Harding, G.C., 1977. Maximizing the minimum source-sink path subject to a budget constraint. Mathematical Programming 13, 116–118.

Furini, F., Ljubić, I., Martin, S., San Segundo, P., 2019. The maximum clique interdiction problem. European Journal of Operational Research 277, 112–127.

Garey, M.R., Johnson, D.S., 1979. Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman.

Ghare, P.M., Montgomery, D.C., Turner, W.C., 1971. Optimal interdiction policy for a flow network. Naval Research Logistics Quarterly 18, 37–45.

Gilbert, E.N., 1959. Random Graphs. The Annals of Mathematical Statistics 30, 1141 – 1144.

Glover, F., 1977. Heuristics for integer programming using surrogate constraints. Decision Sciences 8, 156–166. doi:https://doi.org/10.1111/j.1540-5915.1977.tb01074.x.

Grüne, C., Wulf, L., 2023. A large and natural class of σ_2^p- and σ_3^p-complete problems in bilevel and robust optimization. ArXiv:2311.10540.

Hansen, P., Mladenović, N., 2006. First vs. best improvement: an empirical study. Discrete Applied Mathematics 154, 802–817.

Hoffman, A.J., Kruskal, J.B., 1956. Integral boundary points of convex polyhedra, in: Linear Inequalities and Related Systems. Princeton University Press, pp. 223–246.

Hoos, H.H., Stützle, T., 2004. Stochastic local search: Foundations and applications. Elsevier.

Hosteins, P., 2020. A compact mixed integer linear formulation for safe set problems. Optimization Letters 14, 2127–2148.

Israeli, E., Wood, R.K., 2002. Shortest-path network interdiction. Networks 40, 97–111.

Leitner, M., Ljubić, I., Monaci, M., Sinnl, M., Tanınmış, K., 2023. An exact method for binary fortification games. European Journal of Operational Research 307, 1026–1039.

Lozano, L., Smith, J.C., 2017. A backward sampling framework for interdiction problems with fortification. INFORMS Journal on Computing 29, 123–139.

Lunday, B.J., 2024. The maximal covering location disruption problem. Computers & Operations Research 169, 106721.

Macambira, A.F.U., Simonetti, L., Barbalho, H., Gonzàlez Silva, P.H., Maculan, N., 2019. A new formulation for the safe set problem on graphs. Computers and Operations Research 111, 346–356.

Malaguti, E., Pedrotti, V., 2023. Models and algorithms for the weighted safe set problem. Discrete Applied Mathematics 329, 23–34.

Malik, K., Mittal, A.K., Gupta, S.K., 1989. The k most vital arcs in the shortest path problem. Operations Research Letters 8, 223–227.

Mansi, R., Alves, C., Valério de Carvalho, J.M., Hanafi, S., 2012. An exact algorithm for bilevel 0-1 knapsack problems. Mathematical Problems in Engineering 2012, 504713.

Morton, D.P., Pan, F., Saeger, K.J., 2007. Models for nuclear smuggling interdiction. IIE Transactions 39, 3–14.

PassMark Software, 2022. CPU benchmarks. `https://www.cpubenchmark.net`.

Phillips, C.A., 1993. The network inhibition problem, in: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery. p. 776–785.

Rutenburg, V., 1991. Complexity classification of truth maintenance systems, in: STACS 91: 8th Annual Symposium on Theoretical Aspects of Computer Science Hamburg, Germany, February 14–16, 1991 Proceedings 8, Springer. pp. 372–383.

Scaparra, M.P., Church, R.L., 2008. An exact solution approach for the interdiction median problem with fortification. European Journal of Operational Research 189, 76–92.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: Maher, M., Puget, J.F. (Eds.), Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP '98), Springer–Verlag. pp. 417–431.

Smith, J.C., Song, Y., 2020. A survey of network interdiction models and algorithms. European Journal of Operational Research 283, 797–811.

Wang, Y., Buchanan, A., Butenko, S., 2017. On imposing connectivity constraints in integer programs. Mathematical Programming 166, 241 – 271.

Watts, D.J., Strogatz, S.H., 1998. Collective dynamics of “small-world” networks. Nature 393, 440–442.

Weninger, N., Fukasawa, R., 2023. A fast combinatorial algorithm for the bilevel knapsack problem with interdiction constraints, in: Del Pia, A., Kaibel, V. (Eds.), Integer Programming and Combinatorial Optimization, Springer International Publishing, Cham. pp. 438–452.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics Bulletin 1, 80–83.

Wollmer, R., 1964. Removing arcs from a network. Operations Research 12, 934–940.

Wood, R., 1993. Deterministic network interdiction. Mathematical and Computer Modelling 17, 1–18.