A combinatorial branch and bound for the safe set problem

A combinatorial branch & bound for the Safe Set Problem

Alberto Boggio Tomasaz

Dipartimento di Informatica, Università degli Studi di Milano

Roberto Cordone*

Dipartimento di Informatica, Università degli Studi di Milano

Pierre Hosteins

Université Gustave Eiffel, COSYS-ESTAS, Villeneuve d'Ascq, France

Abstract

The *Weighted Safe Set Problem* requires to partition an undirected graph into two families of connected components, respectively denoted as safe and unsafe, in such a way that each safe component dominates the unsafe adjacent components with respect to a weight function. We introduce a combinatorial branch & 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. The algorithm solves all weighted instances available in the literature and most unweighted ones, up to 50 vertices, with computational times orders of magnitude smaller than the competing algorithms. In order to investigate the limits of the approach, we introduce a benchmark of graphs with 60 vertices, solving to optimality the denser instances.

Keywords: Safe Set, Branch & bound, graph partitioning

1. Introduction

Let $G = (V, E)$ be a connected undirected graph with $n = |V|$ and $w : V \to \mathbb{Q}_+$ a positive weight function defined on its vertices. The weight of any subset of vertices $T \subseteq V$ is defined as the sum of the weights of all vertices in T: $w(T) = \sum_{v \in T} w_v$. We denote as $G[T]$ the subgraph induced on G by the vertices of T and as $\mathcal{C}_G(T)$ the collection of all maximal connected components in the induced subgraph $G[T]$. We generalise the adjacency relation from vertices to disjoint subsets $R, T \subseteq V$ by introducing the relation $R \bowtie_G T$ which holds when there exists an edge between a vertex of R and a vertex of T. The notation is simplified to $\bowtie$ when the graph G is clear from the context.

*Corresponding author

Email addresses: alberto.boggio@unimi.it (Alberto Boggio Tomasaz), roberto.cordone@unimi.it (Roberto Cordone), pierre.hosteins@univ-eiffel.fr (Pierre Hosteins)

A subset $S \subseteq V$ is said to be *safe* when each connected component $S_i \in \mathcal{C}_G(S)$ has a weight not smaller than the weight of any connected component $U_j \in \mathcal{C}_G(V \setminus S)$ such that $U_j \bowtie S_i$. The components of $\mathcal{C}_G(S)$ and $\mathcal{C}_G(V \setminus S)$ are denoted, respectively, as *safe* and *unsafe*. The *Weighted Safe Set Problem* (*WSSP*) requires to find in G a safe set of minimum weight.

The literature discusses an application of the *WSSP* to the design of temporary refuges in buildings, to shelter people from the surrounding spaces in the most volume-efficient way (Fujita et al., 2014). Another application concerns the determination of communities in a social network, whose prevailing weights allow to influence the other communities (Bapat et al., 2016). In a more general fashion, the problem can model the search for a subnetwork whose control guarantees some form of dominance over the entire network.

The early contributions to the study of this problem are theoretically oriented. Fujita et al. (2016) prove its strong $\mathcal{NP}$-hardness on general graphs, even in the unweighted case ($w_v = 1$ for all $v \in V$). For general weight functions, the *WSSP* is $\mathcal{NP}$-hard even on star graphs, but can be solved in $O(n^3)$-time on paths (Bapat et al., 2016). On bounded-treewidth graphs and interval graphs, the problem is only weakly $\mathcal{NP}$-hard, and Àgueda et al. (2018) solve it with pseudopolynomial algorithms based on kernelisation and dynamic programming, that become polynomial in the unweighted case. In particular, trees admit an $O(n^5)$-time algorithm. On planar graphs and split graphs, the problem is $\mathcal{NP}$-hard even in the unweighted case: an approximation-preserving reduction from the Minimum Vertex Cover Problem (Dinur and Safra, 2005) implies that approximation guarantees better than 1.3606 are possible only if $\mathcal{P} = \mathcal{NP}$ (Àgueda et al., 2018). Belmonte et al. (2020) analyse the parameterised complexity of the unweighted problem. The variant of the problem known as *Connected Safe Set Problem* (*CSSP*) imposes on the solution the additional constraint to induce a connected subgraph. It was also introduced in Fujita et al. (2014) and further studied by Ehard and Rautenbach (2020) and Fujita et al. (2020, 2021) with the aim to characterise the maximum gap between the optimal values of the two problems and the instances for which they coincide.

To the best of our knowledge, four papers propose computational approaches to the *WSSP*. Macambira et al. (2019) introduce a randomized stingy heuristic inspired by the algorithm by Fujita et al. (2016) for trees. They also propose an Integer Linear Programming (*ILP*) formulation using $O(n^3)$ binary variables. Since the number of constraints is exponential, they propose a branch & cut approach with separation procedures to generate violated constraints at each branching node (Mitchell, 2009). This approach solves instances up to 30 vertices in a couple of hours. Hosteins (2020) presents a Mixed Integer Linear Programming (*MILP*) formulation using $O(n^2)$ binary variables and $O(n^2)$ real variables. This formulation has polynomial size, as it includes only $O(n^3)$ constraints; in particular, the connectivity requirements are enforced by flow conservation. Experimental results show that it is able to solve instances up to 40 vertices in one hour. Malaguti and Pedrotti (2021, 2022) introduce an *ILP* formulation for the *WSSP* with n variables that indicate the safe vertices and an exponential number of safety constraints. For this noncompact formulation, the approach is again a branch & cut algorithm, that improves upon both Macambira et al. (2019) and Hosteins (2020). All these approaches can be adapted

to the connected version of the problem.

In this paper, we propose a combinatorial branch & bound approach for the *WSSP*, that can also be naturally extended to provide only connected solutions. Its main feature is the design of a bounding procedure that quickly provides an effective lower bound on the objective, without resorting to linear programming. This procedure proves particularly effective on the dense instances, which are the most challenging ones for the formulations proposed by Macambira et al. (2019) and Malaguti and Pedrotti (2021). A second contribution is given by logical reduction procedures that allow to fix vertices of the graph in or out of the optimal solution. These procedures are particularly effective on the sparse instances, which are slightly harder for the formulation proposed by Hosteins (2020). We evaluate the performance of this algorithm on both the available benchmarks and on new instances up to 60 vertices, solving nearly all of them in a limit time of one hour.

In the next section, we recall the *MILP* formulation by Hosteins (2020). Section 3 describes the branch & bound algorithm and Section 4 discusses the computational experiments. Conclusions close the paper.

2. Mathematical programming formulation

We recall here the mathematical programming formulation of Hosteins (2020). The idea behind the model is to introduce a fictitious flow which can only circulate between vertices of the same (safe or unsafe) component and is absorbed equally by each vertex of the graph. Therefore, we need to add to G an artificial vertex 0 adjacent to every other vertex of the graph, which will be the source of this flow. In practice, we will allow a positive amount of flow from vertex 0 to at most one vertex per component, so as to keep track of the component to which each vertex belongs in the solution. Since there are at most $n-1$ components of either type (safe or unsafe), the integer index c ranges in $\{1, ..., n-1\}$.

The model includes the following families of binary variables:

- $x_u = 1$ if vertex u is safe, 0 otherwise;
- $y_{uv} = 1$ if both the extreme vertices of edge (u, v) are safe, 0 otherwise;
- $y'_{uv} = 1$ if both the extreme vertices of edge (u, v) are unsafe, 0 otherwise;
- $a_{uc} = 1$ if vertex u is safe and included in component c;
- $a'_{uc} = 1$ if vertex u is unsafe and included in component c;
- $t_{uc} = 1$ if vertex u is safe, belongs to component c and receives a nonzero flow from vertex 0;
- $t'_{uc} = 1$ if vertex u is unsafe, belongs to component c and receives a nonzero flow from vertex 0;

and of continuous variables:

- f_{uv}: flow from vertex $u \in V \cup \{0\}$ to adjacent vertex $v \in V$;

- ω_u: total weight of the safe component which includes $u \in V$ (0 if u is unsafe);
- ω'_u: total weight of the unsafe component which includes $u \in V$ (0 if u is safe);
- ν_c: total weight of the safe component c;
- and ν'_c: total weight of the unsafe component c.

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

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

and the quantity W is defined as the total weight of the graph:

$$W := \sum_{u \in V} w_u.$$

Consequently, the *WSSP* can be formulated as:

$$\min \sum_{u \in V} w_u \cdot x_u \tag{1a}$$

Subject to:

$$y_{uv} \geq x_u + x_v - 1 \qquad (u, v) \in E \tag{1b}$$

$$y'_{uv} \geq 1 - x_u - x_v \qquad (u, v) \in E \tag{1c}$$

$$y_{uv} \leq x_u \qquad (u, v) \in E \tag{1d}$$

$$y_{uv} \leq x_v \qquad (u, v) \in E \tag{1e}$$

$$y'_{uv} \leq 1 - x_u \qquad (u, v) \in E \tag{1f}$$

$$y'_{uv} \leq 1 - x_v \qquad (u, v) \in E \tag{1g}$$

$$\sum_{v \in N_u} f_{uv} - \sum_{v \in N_u \cup \{0\}} f_{vu} = -1 \qquad u \in V \tag{1h}$$

$$\sum_{v \in V} f_{0v} = n \tag{1i}$$

$$f_{0u} \leq (n-1) \cdot \sum_{c=1}^{n-1} (t_{uc} + t'_{uc}) \qquad u \in V \tag{1j}$$

$$f_{uv} \leq (n-1) \cdot (y_{uv} + y'_{uv}) \qquad (u, v) \in E \tag{1k}$$

$$\sum_{c=1}^{n-1} a_{uc} = x_u \qquad u \in V \tag{1l}$$

$$\sum_{c=1}^{n-1} a'_{uc} = 1 - x_u \qquad u \in V \tag{1m}$$

$$a_{uc} \geq a_{vc} + y_{uv} - 1 \qquad (u,v) \in E, \quad c \in \{1,...,n-1\} \qquad \text{(1n)}$$

$$a_{vc} \geq a_{uc} + y_{uv} - 1 \qquad (u,v) \in E, \quad c \in \{1,...,n-1\} \qquad \text{(1o)}$$

$$a'_{uc} \geq a'_{vc} + y'_{uv} - 1 \qquad (u,v) \in E, \quad c \in \{1,...,n-1\} \qquad \text{(1p)}$$

$$a'_{vc} \geq a'_{uc} + y'_{uv} - 1 \qquad (u,v) \in E, \quad c \in \{1,...,n-1\} \qquad \text{(1q)}$$

$$t_{uc} \leq a_{uc} \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1r)}$$

$$t'_{uc} \leq a'_{uc} \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1s)}$$

$$\sum_{u \in V} t_{uc} \leq 1 \qquad c \in \{1,...,n-1\} \qquad \text{(1t)}$$

$$\sum_{u \in V} t'_{uc} \leq 1 \qquad c \in \{1,...,n-1\} \qquad \text{(1u)}$$

$$\nu_c = \sum_{u \in V} w_u \cdot a_{uc} \qquad c \in \{1,...,n-1\} \qquad \text{(1v)}$$

$$\nu'_c = \sum_{u \in V} w_u \cdot a'_{uc} \qquad c \in \{1,...,n-1\} \qquad \text{(1w)}$$

$$\omega_u \geq \nu_c - W \cdot (1 - a_{uc}) \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1x)}$$

$$\omega_u \leq \nu_c + W \cdot (1 - a_{uc}) \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1y)}$$

$$\omega_u \leq W \cdot x_u \qquad u \in V \qquad \text{(1z)}$$

$$\omega'_u \geq \nu'_c - W \cdot (1 - a'_{uc}) \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1aa)}$$

$$\omega'_u \leq \nu'_c + W \cdot (1 - a'_{uc}) \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1ab)}$$

$$\omega'_u \leq W \cdot (1 - x_u) \qquad u \in V \qquad \text{(1ac)}$$

$$\omega_u \geq \omega'_v - W \cdot y'_{uv} \qquad (u,v) \in E \qquad \text{(1ad)}$$

$$\sum_{u \in V} x_u \geq 1 \qquad \text{(1ae)}$$

$$y_{uv}, y'_{uv} \in \{0,1\} \qquad (u,v) \in E \qquad \text{(1af)}$$

$$a_{uc}, a'_{uc} \in \{0,1\} \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1ag)}$$

$$\omega_u, \omega'_u \geq 0 \qquad u \in V \qquad \text{(1ah)}$$

$$\nu_c, \nu'_c \geq 0 \qquad c \in \{1,...,n-1\} \qquad \text{(1ai)}$$

$$0 \leq f_{uv} \leq n-1 \qquad u \in V \cup \{0\}, \quad v \in V \qquad \text{(1aj)}$$

$$x_u \in \{0,1\} \qquad u \in V \qquad \text{(1ak)}$$

$$t_{uc}, t'_{uc} \in \{0,1\} \qquad u \in V, \quad c \in \{1,...,n-1\} \qquad \text{(1al)}$$

The objective function (1a) is the total weight of the safe vertices. Constraints (1b - 1g) guarantee the consistency of variables y and y' with variables x. They are a linearization of the constraints $y_{uv} = x_u \cdot x_v$ and $y'_{uv} = (1 - x_u) \cdot (1 - x_v)$ for each $(u,v) \in E$. Constraints (1h) force the inflow of a vertex to exceed the

outflow by 1, because every vertex consumes one unit of flow. Constraint (1i) imposes an outflow equal to n from vertex 0. Constraints (1j) impose that a vertex receiving flow from vertex 0 becomes the representative vertex of a component. Constraints (1k) prevent any flow through edges between different components. Constraints (1l, 1m) state that each vertex belongs precisely to one safe or unsafe component. Constraints (1n - 1q) guarantee that vertices of the same component have the same index c. Constraints (1r, 1s) enforce consistency between the components described by variables a and a' and those described by variables t and t'. Constraints (1t, 1u) guarantee that at most one vertex for each component receives flow directly from vertex 0. Constraints (1v, 1w) assign to ν_c the total weight of component c. Constraints (1x - 1ac) assign to ω_u the total weight of the component that contains vertex u. Constraints (1ad) express the safety requirements and, finally, constraint (1ae) imposes that the safe set is not empty.

Hosteins (2020) demonstrates that, if the integrality property holds for variables x, t and t', it is possible to relax the integrality constraint over variables y, y', a and a'. Additional symmetry breaking constraints are provided, along with a variable reduction procedure based on an evaluation of the maximum number of maximal connected components in both the safe and unsafe sets.

3. The branch & bound algorithm

We describe hereafter the scheme of the combinatorial branch & bound algorithm developed to solve the *WSSP*. The algorithm proceeds by fixing vertices in or out of the solution. Each node of the branching tree, therefore, corresponds to a subproblem in which the vertex set V is partitioned into (S, U, F), 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 subproblem $(S, V \setminus S, \emptyset)$ with no free vertices, we can generalise the concepts of safe and unsafe components previously introduced by defining:

$\mathcal{C}_G(S) = \{S_i \mid i = 1, 2, ..., k\}$ as the collection of all safe components S_i (with $k = |\mathcal{C}_G(S)|$),
$\mathcal{C}_G(U) = \{U_j \mid j = 1, 2, ...|\mathcal{C}_G(U)|\}$ as the collection of all unsafe components U_j,
$\mathcal{C}_G(F) = \{F_l \mid l = 1, 2, ..., |\mathcal{C}_G(F)|\}$ as the collection of all free components F_l.

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 subproblem. 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 subproblem (S, U, F) 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 subproblem is fully included in exactly one safe component of the solution, and each unsafe component of the subproblem is fully included in exactly one of the unsafe components of the solution.

Proposition 1. *Let (S, U, F) be a subproblem 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 subproblem. For each component $S_i \in \mathcal{C}_G(S)$ there is a component $\tilde{S}_x \in \mathcal{C}_G(\tilde{S})$ such that $S_i \subseteq \tilde{S}_x$ and $w(S_i) \leq w(\tilde{S}_x)$; for each component $U_j \in \mathcal{C}_G(U)$ there is a component $\tilde{U}_y \in \mathcal{C}_G(V \setminus \tilde{S})$ such that $U_j \subseteq \tilde{U}_y$ and $w(U_j) \leq w(\tilde{U}_y)$.*

Proof. Consider a safe component $S_i \in \mathcal{C}_G(S)$. 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 $\tilde{S}_x \in \mathcal{C}_G(\tilde{S})$. 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 nonnegative, $S_i \subseteq \tilde{S}_x$ 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 subproblem 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 $F_l \in \mathcal{C}_G(F)$ is called* unsavable *if:*

1. *F_l is not adjacent to any safe component;*
2. *its weight $w(F_l)$ 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 subproblem (S, U, F), the vertices of the unsavable free components are unsafe.*

Proof. Let $F_l \in \mathcal{C}_G(F)$ be an unsavable free component, and $U_j \in \mathcal{C}_G(U)$ an unsafe component adjacent to F_l such that $w(U_j) > w(F_l)$. Let $\tilde{S}$ be a feasible solution and $\tilde{U}_j \in \mathcal{C}_G(V \setminus \tilde{S})$ the unsafe component that contains U_j (Proposition 1). Either $\tilde{U}_j$ fully includes also F_l, or there exists a vertex $v \in F_l$ 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 $\tilde{S}^{(v)} \in \mathcal{C}_G(\tilde{S})$ be the safe component that includes v in $\tilde{S}$. Since F_l is unsavable, $\tilde{S}^{(v)} \subseteq F_l$, but all weights are nonnegative, and therefore $w(\tilde{S}^{(v)}) \leq w(F_l) < w(U_j) \leq w(\tilde{U}_j)$, 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 subproblem (S, U, F) 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 subproblem is trivially feasible. If, on the contrary, $S \cup F$ is infeasible, there is a maximal connected component $\hat{S}_i \in \mathcal{C}_G(S \cup F)$ that is adjacent to a maximal connected component $U_j \in \mathcal{C}_G(U)$, 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 $\tilde{U}_y \in \mathcal{C}_G(V \setminus \tilde{S})$. 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 subproblems 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 subproblem (S, U, F). 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 $\mathcal{C}_G(U)$ provides a lower bound on the optimum: $w(\tilde{S}) \geq \max_{U_j \in \mathcal{C}_G(U)} w(U_j)$.

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 paper and the main reason for the effectiveness of the branch & 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 1 shows a subproblem (S, U, F) in which $S = \{1, 6\}$, $U = \{3, 7\}$ and $F = \{2, 4, 5\}$. The 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 subproblem, $S^* = \{1, 4, 5, 6\}$.

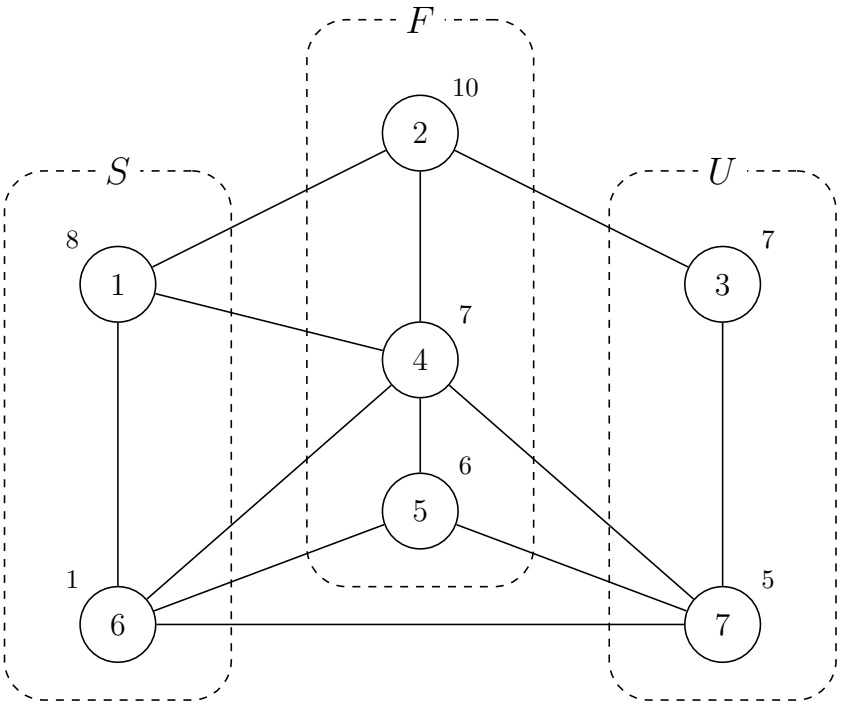

Figure 1: Example of a subproblem (S, U, F). The weight of each node is displayed next to it.

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 subproblem (S, U, F), 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 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 $\mathcal{NP}$-complete problem is hard to solve.

In order to deal with all these aspects, we assume that S and U are not empty, and that at most one of the $|\mathcal{C}_G(S)| = k \geq 1$ components of $\mathcal{C}_G(S)$ is adjacent to F. The first limitation is true in most subproblems, 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 $i = 1, \ldots, k-1$), S_k possibly incorporates free vertices ($\tilde{S}_k \supseteq S_k$), and new safe components $\tilde{S}_i$ ($i = k+1, \ldots, r$) are possibly created from scratch. Figure 2 provides examples of the possible cases. Given subproblem (S, U, F) 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 subproblem (S, U, F) guarantees that none

of the safe components of S merge.

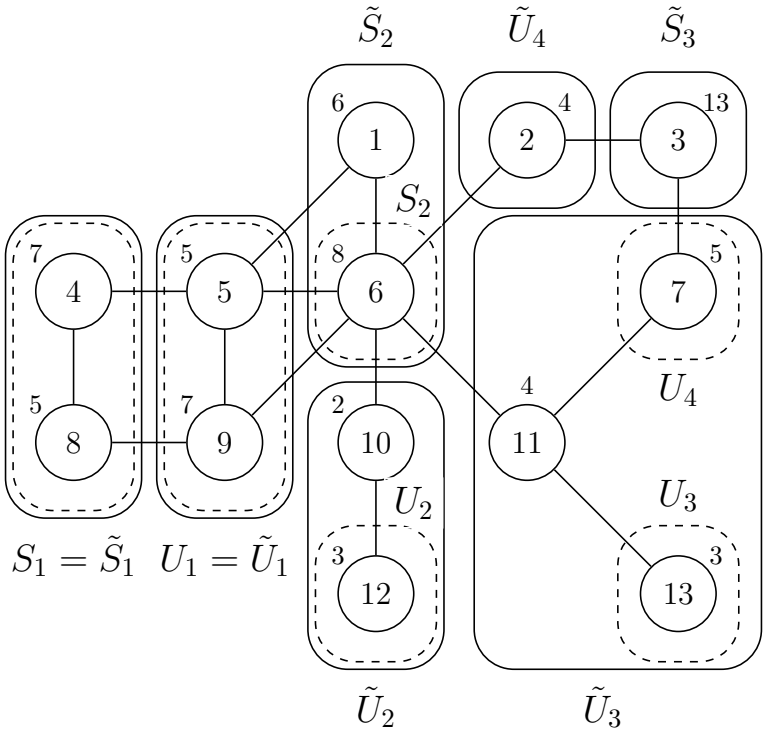

Figure 2: The safe and unsafe components of subproblem (S, U, F) (marked in dashed lines) satisfy two basic limitations: S and U are nonempty, and at most one of the safe components of $\mathcal{C}_G(S)$, namely S_2, is adjacent to F. Considering any feasible solutions of the subproblem (e.g, $\tilde{S} = \{1, 3, 4, 6, 8\}$), 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 ($\tilde{U}_3 = U_3 \cup \{11\} \cup U_4$).

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 subproblem. 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 subproblem (S, U, F) 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 nonadjacent 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 nonconnected 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 $s \in S_k$ provides a relaxation of subproblem (S, U, F).*

Proof. Consider any feasible solution $\tilde{S}$ and let (u, v) be an edge with $u \in F'$ and $v \in F \setminus F'$. If $v \in \tilde{S}$, the safe component of $\tilde{S}$ that includes v merges with S_k (unless they already were the same), the weight of their union is larger than the original weights and all safety constraints are even more 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 shows an example of these relaxations for graph $G = (V, E)$ with $V = \{1, \ldots, 13\}$ and the subproblem 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 1 provides examples of all the modifications discussed in the previous propositions on the graph of Figure 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

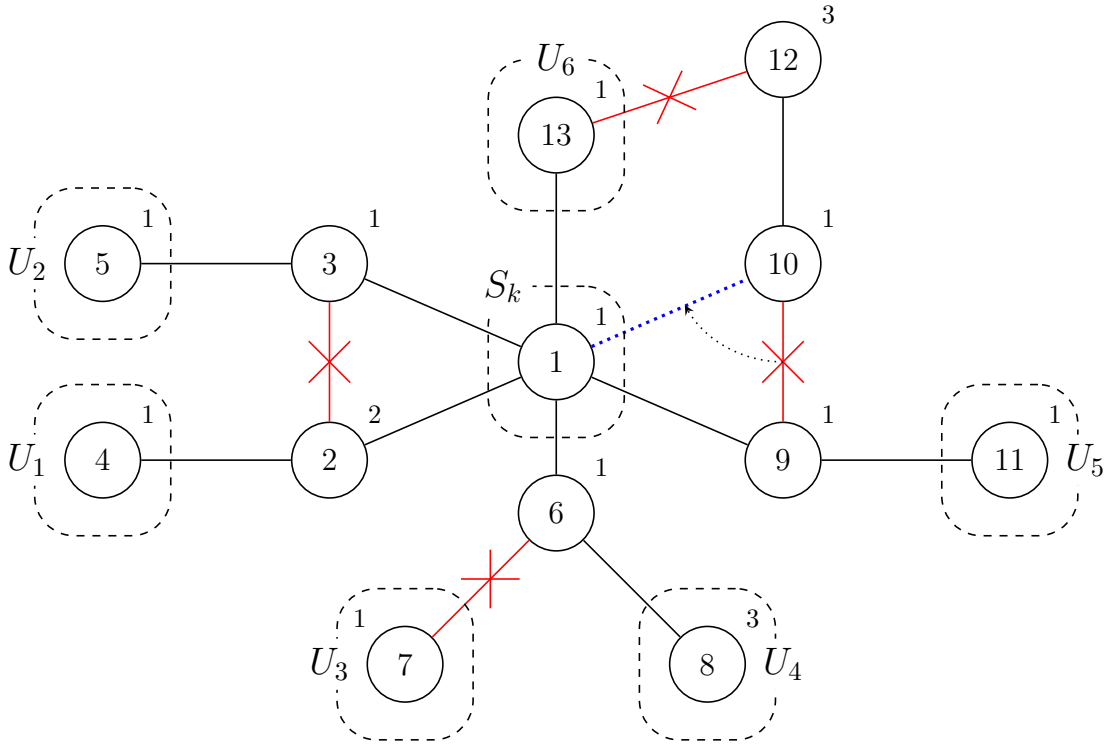

Figure 3: Example of graph manipulations: removing each of the three edges $(2,3)$, $(6,7)$ or $(12,13)$ (in red) or replacing edge $(9,10)$ (also in red) with $(1,10)$ (in dotted blue) yields a relaxation of the original problem.

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 1: Manipulations applied to the graph in Figure 3 and their effects on the optimal solution.

The lower bounding procedure adopted in the branch & bound algorithm 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 $U_l \in \mathcal{C}_G(U)$ 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 $\tilde{U}_l \in \mathcal{C}_G(V \setminus \tilde{S})$. 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 maximum weight adjacent unsafe component, obtaining a partition of F' into subsets associated

to the unsafe components U_l.

$$F'_l = \left\{ v \in F' \mid U_l = \underset{U_y \bowtie_G \{v\}}{\arg\max}\, w(U_y) \right\} \qquad l \in L',$$

where $L' \subseteq \{1, \ldots, |\mathcal{C}_G(U)|\}$ 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 $\tilde{U}_l$ 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 1 to Figure 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 subproblem (S, U, F) on graph G' must satisfy the safety constraints

$$w(\tilde{S}_i) \geq w(\tilde{U}_l) \qquad \tilde{S}_i \in \mathcal{C}_{G'}(\tilde{S}), \tilde{U}_l \in \mathcal{C}_{G'}(V \setminus \tilde{S}) : \tilde{S}_i \bowtie_{G'} \tilde{U}_l, \tag{2}$$

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

$$w(\tilde{S}_k) = w(S_k) + \sum_{v \in F \cap \tilde{S}_k} w_v x_v \geq w(U_l) + \sum_{v \in F'_l} w_v (1 - x_v) = w(\tilde{U}_l) \quad l \in L', \tag{3}$$

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:

$$\min w(S) + \sum_{v \in F} w_v x_v \tag{4a}$$

$$w(S_k) + \sum_{v \in F} w_v x_v \geq w(U_l) + \sum_{v \in F'_l} w_v (1 - x_v) \qquad l \in L' \tag{4b}$$

$$x_v \in \{0, 1\} \qquad v \in F. \tag{4c}$$

This problem generalises the optimisation version of the *Subset Sum problem*, that is $\mathcal{NP}$-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 (4) 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 $\tau_l = \sum_{v \in F'_l} w_v (1 - x_v)$ with $l \in L'$. These variables

satisfy the following properties:

$$\tau_l = \sum_{v \in F'_l} w_v(1 - x_v) \leq \sum_{v \in F'_l} w_v = w(F'_l) \qquad l \in L' \tag{5}$$

$$\begin{aligned} \sigma + \sum_{l \in L'} \tau_l &= \sum_{v \in F \setminus F'} w_v x_v + \sum_{v \in F'} w_v x_v + \sum_{l \in L'} \sum_{v \in F'_l} w_v(1 - x_v) \\ &= \sum_{v \in F \setminus F'} w_v x_v + \sum_{v \in F'} w_v x_v + \sum_{v \in F'} w_v(1 - x_v) \\ &\geq w(F') \end{aligned} \tag{6}$$

from which we derive

$$\min \sigma \tag{7a}$$

$$w(S_k) + \sigma \geq w(\tilde{U}_l) + \tau_l \quad l \in L' \tag{7b}$$

$$\tau_l \leq w(F'_l) \qquad l \in L' \tag{7c}$$

$$\sigma + \sum_{l \in L'} \tau_l \geq w(F') \tag{7d}$$

$$\sigma, \tau_l \geq 0 \qquad l \in L' \tag{7e}$$

To solve problem (7) and compute a lower bound for subproblem (S, U, F), we propose Algorithm 1. First, by setting $\tau_l = 0$ for all $l \in L'$ and $\sigma = \max\{0, \max_{l \in L'} w(U_l) - w(S_k)\}$, we obtain the best solution that satisfies all constraints but (7d). Then, we increase each τ_l variable to reduce the infeasibility of (7d), without violating constraints (7b, 7c). The main loop of the algorithm iteratively computes the violation ϕ of constraint (7d) and identifies the unsafe components which still have a residual capacity, 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 $\tau_l < w(F'_l)$) by μ to keep constraints (7c) satisfied. Increasing them by the same amount, also constraints (7b) 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 (7d) was already satisfied before the loop, it means that we already have the optimal solution and that we must not decrease the variables by $\bar{\phi} \leq 0$ because it could possibly violate constraints (7e).

At each iteration, at least one of constraints (7c) is activated, implying that, after at most $|L'|$ iterations, $Q = \emptyset$ and therefore $\mu = +\infty \geq \bar{\phi}$, exiting the loop. Once outside the loop, $\bar{\phi} \leq \mu$, so $\tau_l + \bar{\phi} \leq w(F'_l)$ for all $l \in Q$: it is possible to increase σ and every τ_l with $l \in Q$ by $\bar{\phi}$. Increasing $|Q| + 1$ variables by $\bar{\phi} = \phi/(|Q| + 1)$ reduces the violation of (7d) to zero, thus producing a feasible solution. The optimality of this solution is given by the fact that at each step we increase every variable (which can be feasibly increased) by the same amount, minimising the maximum quantity among $w(S_k) + \sigma$ and $w(U_l) + \tau_l \quad \forall l \in L'$.

$z := \max\left\{w(S_k),\ \max_{l \in L'} w(U_l)\right\}$

$\sigma := z - w(S_k)$ // Satisfies constraints (7b)

$\tau_l := \min\{w(F'_l),\ z - w(U_l)\} \quad \forall l \in L'$ // Keep satisfied constraints (7c)

loop

  $\phi := w(F') - \sigma - \sum_{l \in L'} \tau_l$ // ϕ is the violation of constraint (7d)

  $Q := \{l \in L' \mid \tau_l < w(F'_l)\}$ // Set of all components with residual capacity

  $\bar{\phi} := \phi / (|Q| + 1)$

  $\mu := \min_{l \in Q} \{w(F'_l) - \tau_l\}$ // Minimal residual capacity among Q ($\mu = +\infty$ if $Q = \emptyset$)

  if $\bar{\phi} \leq \mu$ **then exit loop**

  $\tau_l := \tau_l + \mu \quad \forall l \in Q$ // Since $\mu \leq w(F'_l) - \tau_l \quad \forall l \in Q$, (7c) are satisfied

  $\sigma := \sigma + \mu$ // Since the increment is the same, (7b) are satisfied

end

if $\phi > 0$ **then** // Fails only if (7d) were already satisfied before the loop

  $\sigma := \sigma + \bar{\phi}$

  $\tau_l := \tau_l + \bar{\phi} \quad \forall l \in Q$ // $\bar{\phi} \leq \mu$ so the increment respects (7c)

end

return $\sigma, \quad \forall l \in L' : \ \tau_l$

Algorithm 1: Algorithm to solve problem (7)

In Figure 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 $\phi = w(F') - \sigma - \sum_l \tau_l = 29 - 7 - 5 - 2 = 15$ 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 $l \in Q$ 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.

3.3. Vertex fixing

Section 3.1 described a feasibility test for a given subproblem (S, U, F), 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 subproblem. The first one simply extends the feasibility test with an implicit branching operation.

Proposition 5. *Given a subproblem (S, U, F), let C be a component of $\mathcal{C}_G(S \cup F)$ and f a vertex of $C \cap F$. If*

$$w(C) - w_f < \sum_{\substack{U_j \in \mathcal{C}_G(U): \\ U_j \bowtie \{f\}}} w(U_j) + w_f \tag{8}$$

then vertex f belongs to all feasible solutions of (S, U, F).

Proof. Moving f from F to U modifies subproblem (S, U, F) into subproblem $(S, U \cup \{f\}, F \setminus \{f\})$. Correspondingly, all the unsafe components $U_j \in \mathcal{C}_G(U)$ that were adjacent to f ($U_j \bowtie \{f\}$) merge into a single one, whose weight is the right-hand-side of (8). 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 (8). 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 6. *Let (S, U, F) be a subproblem and $f \in F$ a free vertex. If*

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

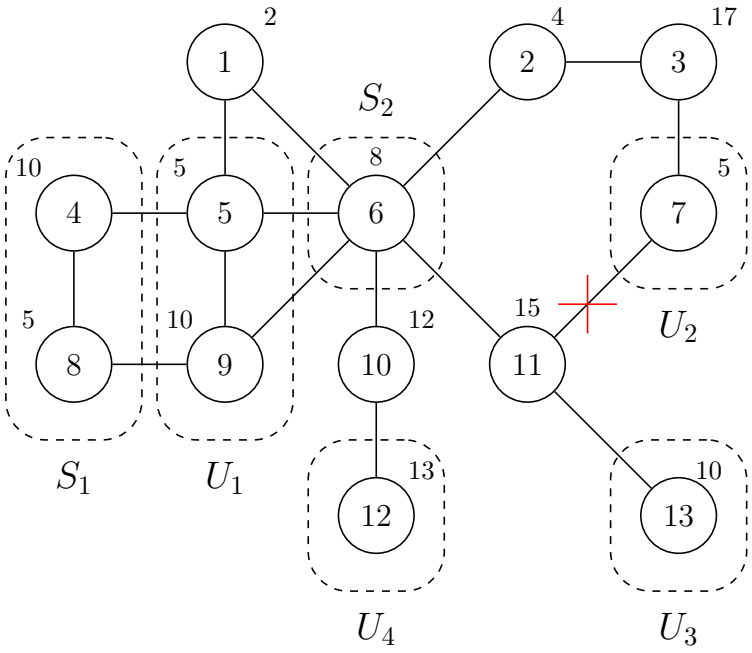

Figure 4: Example of the computation of the refined lower bound for a subproblem (S, U, F): the current total weight of the safe components is $w(S) = 23$ and the current maximum weight of the unsafe components is $w(U_1) = 15$, but the lower bound can be raised up to 35, partitioning $F' = \{1, 10, 11\}$ 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.

then vertex f does not belong to any feasible solution better than $\bar{S}$. If

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

then vertex f belongs to 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 that at most one safe component is adjacent to F. Since this assumption is fundamental to refine the lower bound, if we are applying the bounding procedure, we move into S only free vertices that are already adjacent to S.

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 refined 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 *CSSP*. 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.

4. Experimental results

4.1. Experimental setting

The literature on the *WSSP* provides two sets of benchmark instances. The set used by Macambira et al. (2019) is publicly available at `http://www.cos.ufrj.br/~luidi/papers/safeset.html`. It consists of three

classes, based on the density of graph G: the number of edges is set to $\lfloor \delta n(n-1)/2 \rfloor$, with $\delta \in \{0.3, 0.5, 0.7\}$. Each class includes one instance for each number of vertices n, ranging from 10 to 30. There are weighted and unweighted instances, that consider the same graphs. The weights for the former are randomly extracted from a uniform distribution in $\{1, \ldots, 100\}$[1]. Overall, therefore, this benchmark consists of $3 \cdot 21 \cdot 2 = 126$ instances.

The instances introduced by Hosteins (2020) have a similar structure, but they are partitioned into four classes, with $\delta \in \{0.1, 0.2, 0.3, 0.4\}$, and their number of vertices is $n \in \{20, 25, 30, 35, 40, 50\}$. Moreover, each size corresponds to 5 instances. The benchmark includes weighted and unweighted instances, where the former have weights uniformly extracted from $\{1, \ldots, 10\}$. Also in this case, weighted and unweighted instances share the same graph topology. Overall, this yields $4 \cdot 6 \cdot 5 \cdot 2 = 240$ instances. Since the branch & bound algorithm solves nearly all the existing instances to optimality, we have extended the latter benchmark generating other $4 \cdot 5 \cdot 2 = 40$ instances with $n = 60$ vertices. All the instances and the detailed results are available in the supplementary material and at `https://homes.di.unimi.it/cordone/research/wssp.html`.

The experiments have been conducted on a Linux server, with processor Intel Xeon E5-2620 2.1 GHz and 16 GB of RAM. The algorithm is coded in C99 and compiled with GNU GCC 8.3.0, and runs in a single thread.

4.2. Contribution of the bounding and the vertex fixing procedures

A first phase of experiments has been dedicated to estimate the role played in the performance of the algorithm by the lower bounding procedure discussed in Section 3.2 and the vertex fixing procedures presented in Section 3.3. To this purpose, we have compared four variants of the branch & bound algorithm, that apply, respectively:

- the best among the refined lower bound and the two simple ones, and all vertex fixing procedures (RF);
- the best among the refined lower bound and the two simple ones, with no vertex fixing (RN);
- the better of the two simple lower bounds and all vertex fixing procedures (SF);
- the better of the two simple lower bounds with no vertex fixing (SN).

We remind that RF and RN must respect the limitations discussed at the end of Sections 3.3 and 3.4 when choosing the branching vertex and fixing vertices to safe.

We compared the four versions on the instances from 20 to 40 vertices in the second benchmark, because the first benchmark proved rather easy to solve and the larger instances were too hard for some versions. Figures 5 and 6 report the average computational time required by the four versions of the algorithm to solve to optimality these instances as a function of the number of vertices n. The scale is logarithmic on the time axis

[1] The weighted instances with 18 and 23 vertices have one or two vertices of zero weight. Since the corresponding unweighted instances provide the same graphs and positive weights, we have used the latter as a replacement. This explains some small differences in the published results.

to improve the quality of the visualisation. Figure 5 concerns the weighted instances, while Figure 6 concerns the unweighted ones. Each figure contains four graphs, corresponding to the different values of the density. The two versions with all vertex fixing procedures (in continuous lines) consistently perform better than the other two (in dashed lines), although the difference is marked for the sparse instances and rather limited for the dense ones. This suggests that the computational effort they require is compensated by their effectiveness. As for the bounding procedure, the refined one has a disruptive effect on the dense instances, for which the computational time becomes orders of magnitude lower, whereas the simple procedure performs better on the sparser instances, that is up to $\delta = 0.2$ for the weighted benchmark and $\delta = 0.1$ for the unweighted one. This is consistent with Remark 1, since denser instances have a larger set of free vertices that can contribute to increase the bound. For example, if $|V| = 40$ and $|U| = |S| = 4$, the expected number of free vertices adjacent to both is only $\mathbb{E}[|X|] \approx 5$ out of $|F| = 32$ free vertices when the density is $\delta = 0.1$, but rises to $\mathbb{E}[|X|] \approx 15$ when the density is $\delta = 0.2$ and up to $\mathbb{E}[|X|] \approx 30$ when the density is $\delta = 0.4$. This suggests that, even if the refined bound always dominates the simple one, the effort required to compute it is justified only for sufficiently dense graphs. Consequently, in the following experiments, we decided to apply the vertex fixing procedures, but to adopt the refined bound when $\delta \geq 0.2$ and the simple bound when $\delta < 0.2$.

4.3. Comparison with the state of the art

In the second phase of experiments, we have compared the results of our combinatorial branch & bound algorithm with those of the competing algorithms available in the literature. All these approaches run on different, but comparable, machines. Table 2 reports the acronyms used in the following to denote them and their main features. The number of threads 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 SCIP solver 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.

Acronym	Reference	Processor	Frequency	RAM	Threads
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 (2021, 2022)	Intel Core i7-4790	3.60 GHz	32 GB	1
B&B	This contribution	Intel Xeon E5-2620	2.1 GHz	16 GB	1

Table 2: Acronyms of the existing algorithms for the *WSSP* and machines used to test them

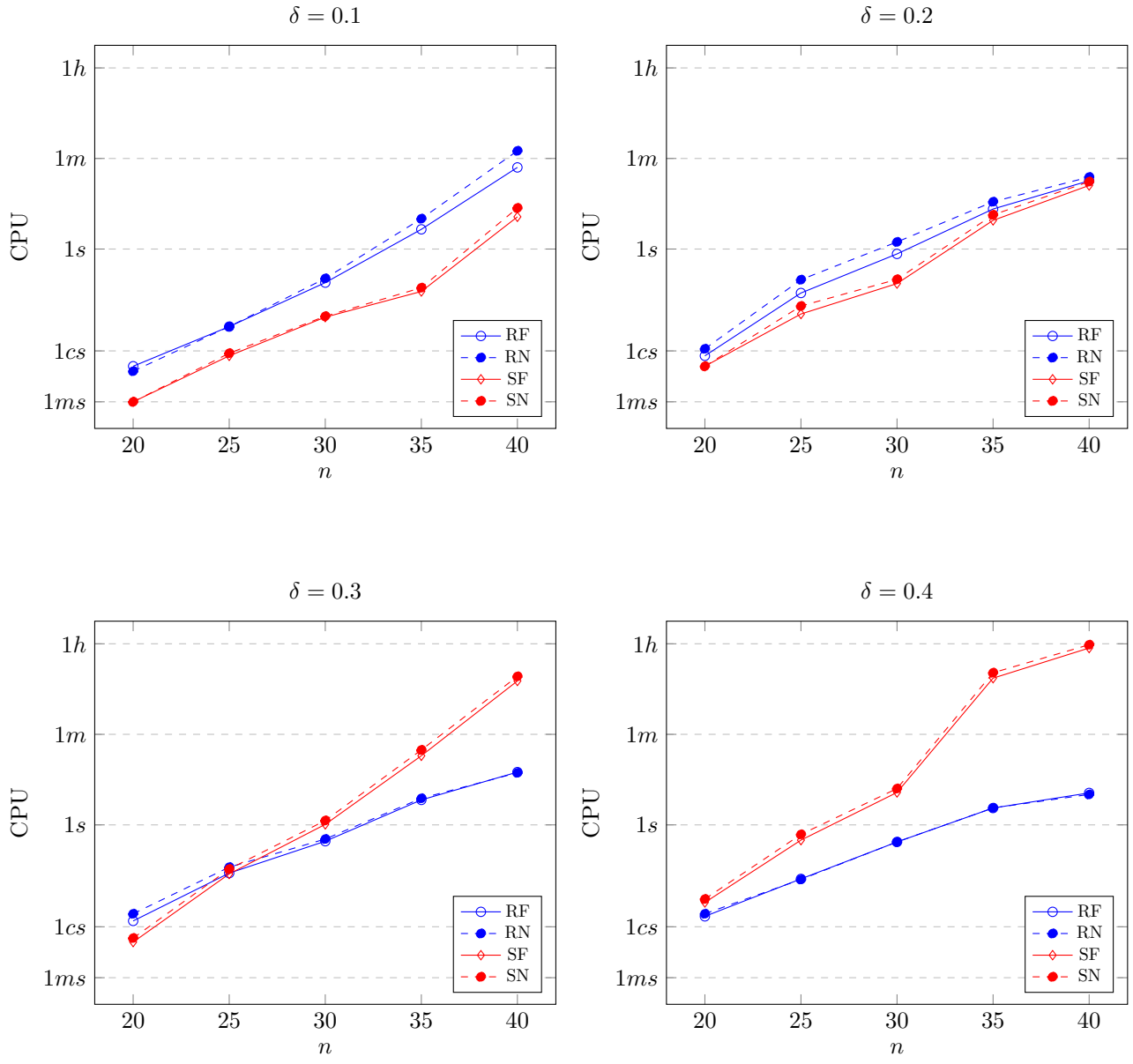

Figure 5: Computational time required by the branch & bound algorithm on the weighted instances of Hosteins (2020), respectively with the refined bound and all fixing procedures (RF), the refined bound and no fixing procedures (RN), the simple bounds and all fixing procedures (SF), the simple bounds and no fixing procedures (SN)

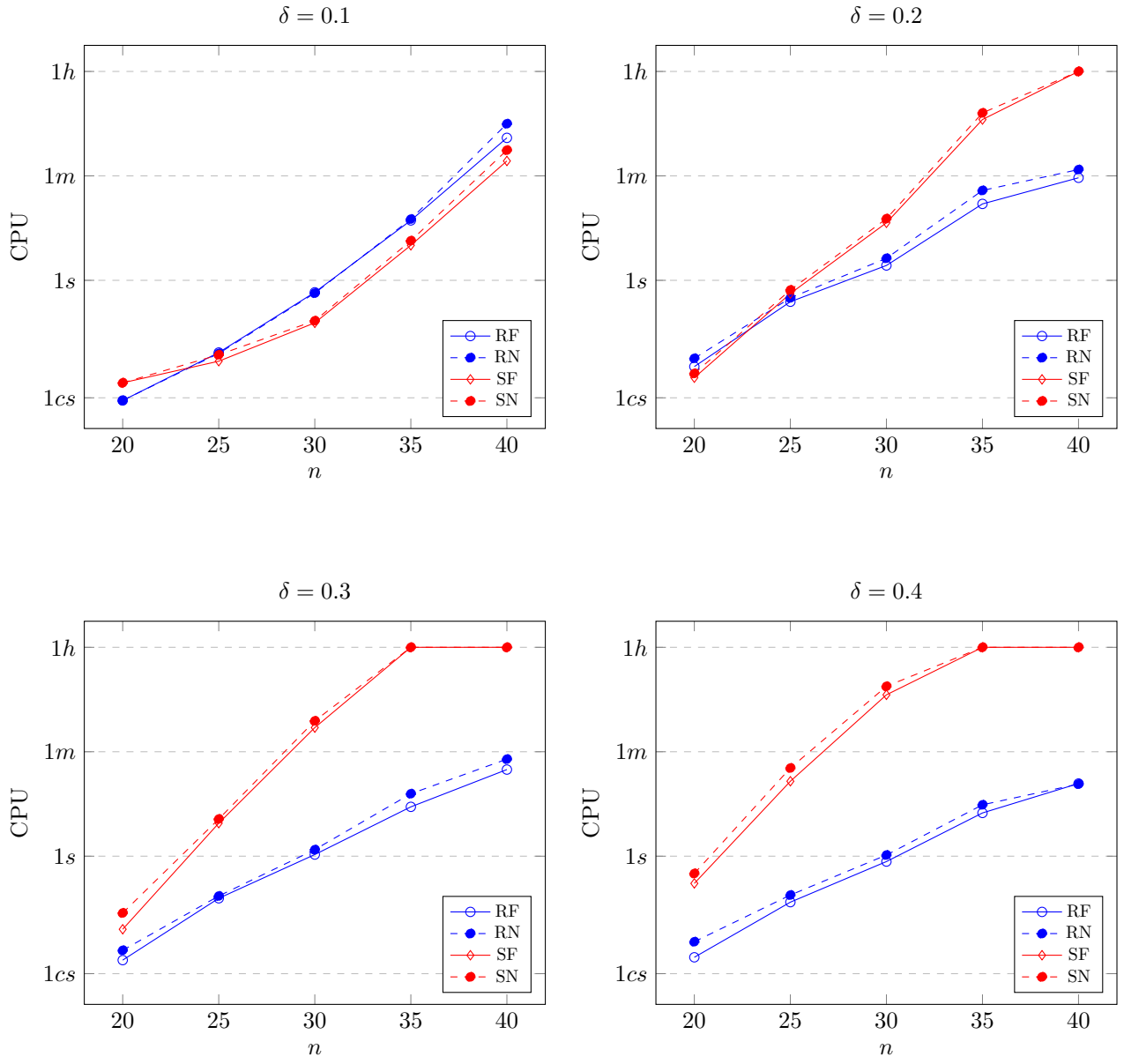

Figure 6: Computational time required by the branch & bound algorithm on the unweighted instances of Hosteins (2020), respectively with the refined bound and all fixing procedures (RF), the refined bound and no fixing procedures (RN), the simple bounds and all fixing procedures (SF), the simple bounds and no fixing procedures (SN); a time limit of one hour is imposed on the execution

Table 3 reports the results for the weighted benchmark introduced by Macambira et al. (2019). 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 7 200 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&B 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.

	$\delta = 0.3$				$\delta = 0.5$				$\delta = 0.7$			
n	B&C1	MILP	B&C2	B&B	B&C1	MILP	B&C2	B&B	B&C1	MILP	B&C2	B&B
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: Computational times (in seconds) required to solve the weighted instances of Macambira et al. (2019) of density $\delta \in \{0.3, 0.5, 0.7\}$ with B&C1, MILP, B&C2 and B&B

Table 4 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, except for the fact that the difference between B&C2 and B&B are more marked (in favour of B&C2 for $\delta = 0.3$ and of B&B 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.

	$\delta = 0.3$				$\delta = 0.5$				$\delta = 0.7$			
n	B&C1	MILP	B&C2	B&B	B&C1	MILP	B&C2	B&B	B&C1	MILP	B&C2	B&B
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 4: Computational times (in seconds) required to solve the unweighted instances of Macambira et al. (2019) of density $\delta \in \{0.3, 0.5, 0.7\}$ with B&C1, MILP, B&C2 and B&B

Table 5 compares the results of MILP, B&C2 and B&B on the benchmark introduced by Hosteins (2020) for the weighted instances. 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 (2022), 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 & 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.

The results for the unweighted instances of the same benchmark are displayed in Table 6 and present similar features. B&B 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.

4.4. A new benchmark of large instances

In order to check the practical limits of the proposed approach, we have then applied it to a new set of instances with 60 vertices, generated as in Hosteins (2020). Table 7 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&B 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 7 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).

4.5. Analysis of the algorithm

This section analyses in detail some quantitative aspects of the algorithm that influence its performance. All results refer to the benchmark proposed by Hosteins (2020) with our extension.

First, we measure how the overall computational time is distributed among the main elements that compose the algorithm. These are the computation of the lower bound, the variable fixings and the generation of the children nodes. The variable fixings are inextricably combined with the feasibility test and with the computation of the upper bound, because they either prove the infeasibility of the current subproblem or provide a feasible solution composed by all the vertices that are not fixed to unsafe. The generation of the children nodes includes the choice of the branching vertex and the construction of the data structure for each new node. We focus on the instances with 50 vertices, the largest ones that in general can be solved to optimality within the time limit,

		MILP			B&C2			B&B		
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 5: Average gap (%), number of instance solved within an hour and average computational time (in seconds) required to solve the weighted instances of Hosteins (2020) of density $\delta \in \{0.1, 0.2, 0.3, 0.4\}$ with MILP, B&C2 and B&B.

		MILP			B&C2			B&B		
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 6: Average gap (%), number of instance solved within an hour and average computational time (in seconds) required to solve the unweighted instances of Hosteins (2020) of density $\delta \in \{0.1, 0.2, 0.3, 0.4\}$ with MILP, B&C2 and B&B.

Instance		Weighted				Unweighted			
δ	#	UB	LB	sec	BN	UB	LB	sec	BN
	1	**77**	**77**	3474.908	214559949	31	13	1405.774	OM
	2	**79**	**79**	637.409	36033589	34	14	1741.854	OM
0.1	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
	1	124	115	3600.000	114519193	30	24	3600.000	124741717
	2	**111**	**111**	818.154	26933287	30	25	3600.000	122956349
0.2	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
	1	**129**	**129**	849.539	20954933	30	27	3600.000	89251817
	2	**124**	**124**	500.606	12054917	30	27	3600.000	90466873
0.3	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
	1	**137**	**137**	169.997	3122629	**29**	**29**	2178.099	40799191
	2	**130**	**130**	236.432	4292569	**29**	**29**	3097.001	60593849
0.4	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 7: Upper bound, lower bound, time (in seconds) and number of branching nodes visited by B&B to solve unweighted and weighted instances of size $|V| = 60$ and density $\delta \in \{0.1, 0.2, 0.3, 0.4\}$.

and we register the fraction of time required by the three procedures, with respect to the overall time. When we adopt the refined lower bound (density $\delta \geq 0.2$), the computation of the lower bound takes about $40-50\%$ of the time, the variable fixings account for $40-45\%$ of the time, the generation of the children nodes is below 5% and the residual operations around $5-10\%$. Using the simple bound decreases the fraction of time for the lower bound to $25-35\%$, increasing those of the variable fixings to $35-50\%$, of the node generation to slightly above 5% and of the residual operations to $10-30\%$. This is not surprising, as the two main procedures involve more or less complex visits of the graph, whereas the other ones consist of simpler operations.

A second aspect of interest is the time required by the upper and the lower bound to converge to their final values. In fact, the optimal value could be approximated in short time, or only at the end of the computation, and the behaviour of the upper and the lower bound in this respect could be different. Figure 7 shows the profile of the upper and lower bound on four instances selected among the largest ones (60 vertices). The two instances on the left are weighted, the two on the right are unweighted. The upper instances have density $\delta = 0.2$, and show a residual gap at the end of the time limit of one hour. The lower instances have density $\delta = 0.4$, and can be solved to optimality, but are among the slowest to be solved. Notice that the time axis is in logarithmic scale to stress the progress in the early phases of the computation. We can observe that in all four plots, the upper bound decreases almost instantaneously, obtaining heuristic solutions which are close to the optimum, though in these hard instances better solutions are found also later on. The lower bound converges more slowly, but it gives a good approximation of the optimum after a minute, and then steadily increases until the end.

Finally, the size of the branching tree is a relevant index of the computational effort required. Figure 8 reports the average number of branching nodes with respect to the size of the graphs for the weighted instances (on the left) and the unweighted ones (on the right), for the four classes of density. In both figures, the graphs are approximately linear in a logarithmic scale, which means that the number of branching nodes grows exponentially with the size of the problem. There is a minor dependency on density, with sparse graphs tending to require more nodes, consistently with the remark that the lower bound becomes weaker. For the weighted instances, the number of branching nodes is approximately one order of magnitude lower than for the unweighted ones. Indeed, the values are missing for the unweighted instances of 50 and 60 vertices with density $\delta = 0.1$ because the computation is prematurely terminated for insufficient memory: they would be in the order of $10^8 - 10^9$.

5. Conclusions

We have proposed a combinatorial branch & bound algorithm for the *Weighted Safe Set Problem*. The algorithm exploits reduction procedures to force vertices in and out of the solution, based on the safety constraints and on the comparison with the value of the best known heuristic solution. Its main feature is a refined lower bounding technique that estimates the contribution of the still unfixed vertices to the safe and unsafe components of any feasible solution, and therefore to the value of the objective function. Computational experiments

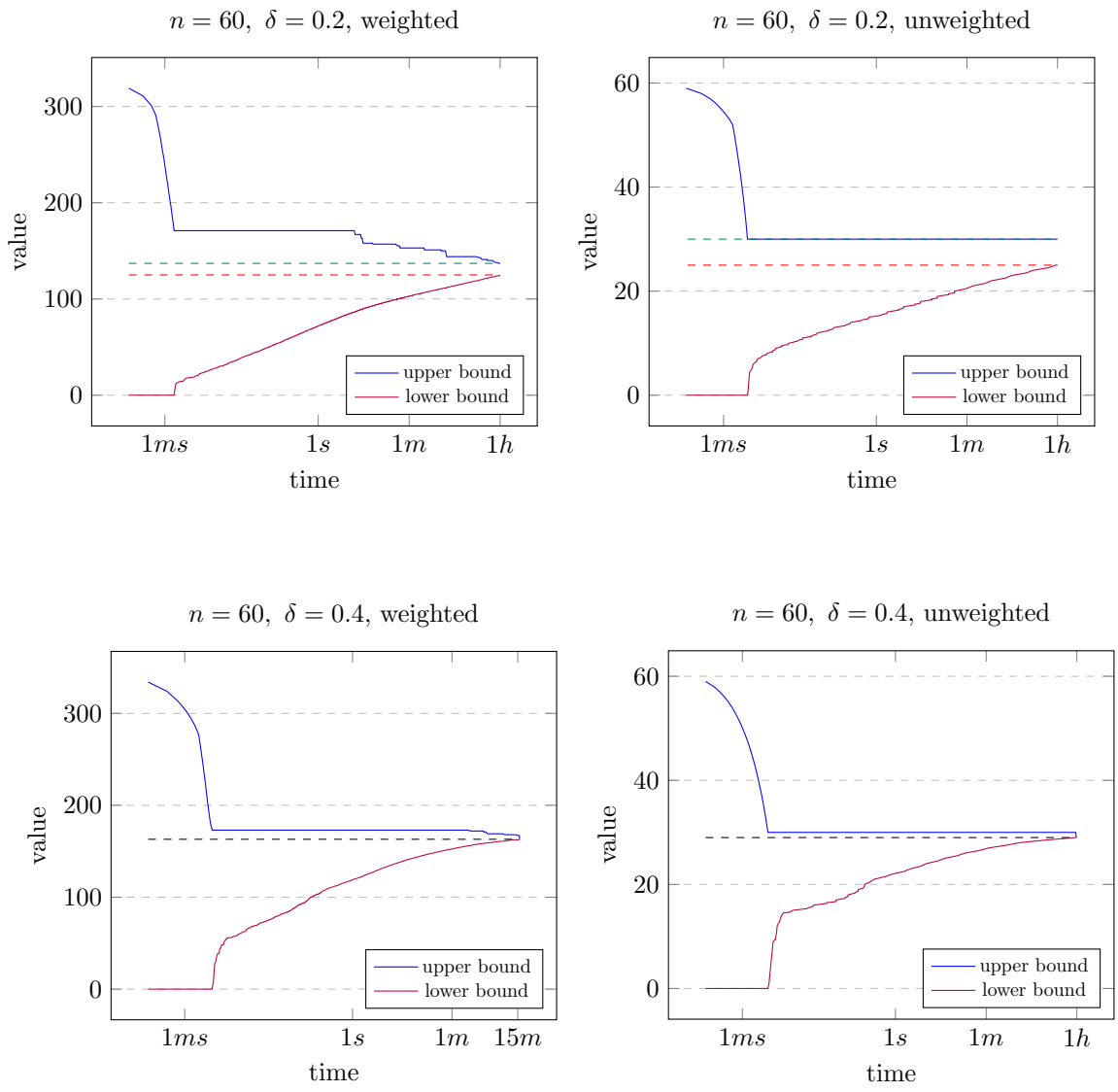

Figure 7: Profiles of the upper bound and the lower bound during the computation on four instances with 60 vertices: the upper row concerns the weighted and unweighted instance `60_p0.2_5`, with density $\delta = 0.2$; the lower row concerns the weighted and unweighted instance `60_p0.4_3`, with density $\delta = 0.4$.

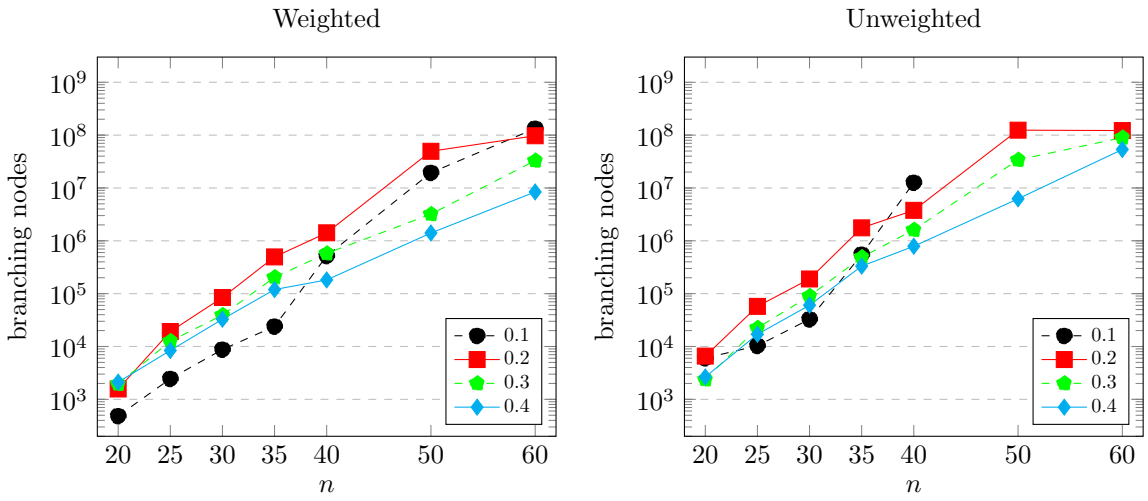

Figure 8: Average number of branching nodes visited by the branch & bound procedure on the benchmark introduced by Hosteins (2020), depending on the number of vertices n and the density of the graph: weighted instances are on the left, unweighted ones on the right.

on the available benchmarks and on newly generated instances show that the algorithm here proposed is the only one able to consistently solve instances of 50 and even 60 vertices. Due to the lower bounding procedure, it is particularly effective on dense instances, whereas on the sparsest ones its performance is comparable, but probably slower (to the extent that the different machines used are interchangeable) than that of the branch & cut approach of Malaguti and Pedrotti (2022). In general, the unweighted instances appear to be harder for all approaches.

References

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.

Bapat, R., Fujita, S., Legay, S., Manoussakis, Y., Matsui, Y., Sakuma, T., Tuza, Z., 2016. Network majority on tree topological network. Electronic Notes in Discrete Mathematics 54, 79–84.

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.

Dinur, I., Safra, S., 2005. On the hardness of approximating minimum vertex cover. Annals of Mathematics 162, 439–485.

Ehard, S., Rautenbach, D., 2020. Approximating connected safe sets in weighted trees. Discrete Applied Mathematics 281, 216–223.

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.

Fujita, S., Park, B., Sakuma, T., 2020. Stable structure on safe set problems in vertex-weighted graphs II - Recognition and complexity, in: Adler, I., Müller, H. (Eds.), Proceedings of the 46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2020), Leeds, UK. pp. 364–375.

Fujita, S., Park, B., Sakuma, T., 2021. Stable structure on safe set problems in vertex-weighted graphs. European Journal of Combinatorics 91, 103211.

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.

Gilbert, E.N., 1959. Random Graphs. The Annals of Mathematical Statistics 30, 1141 – 1144.

Hosteins, P., 2020. A compact mixed integer linear formulation for safe set problems. Optimization Letters 14, 2127—2148.

Macambira, A.F.U., Simonetti, L., Barbalho, H., Silva, P.H.G., Maculan, N., 2019. A new formulation for the safe set problem on graphs. Computers and Operations Research 111, 346–356.

Malaguti, E., Pedrotti, V., 2021. A new formulation for the weighted safe set problem. Procedia Computer Science 195, 508–515.

Malaguti, E., Pedrotti, V., 2022. Models and algorithms for the weighted safe set problem. Optimization Online February 2022. URL: www.optimization-online.org/DB_HTML/2022/02/8783.html.

Mitchell, J.E., 2009. Integer programming: branch and cut algorithms, in: Floudas, C.A., Pardalos, P.M. (Eds.), Encyclopedia of Optimization, Springer US, Boston, MA. pp. 1643–1650.