

# A Revisited Branch and Bound Method for the Weighted Safe Set Problem

Alberto Boggio Tomasaz<sup>a</sup> and Roberto Cordone<sup>b</sup>

Department of Computer Science, University of Milan, Italy

Keywords: Weighted Safe Set Problem, Branch and Bound.

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 some improvements to an existing exact approach that produce a significant reduction in the effort required to find the optimum or in the gap between the optimum and the best solution obtained within a given time limit. The first improvement consists of a relaxation that is weaker than the original one, but allows to adopt a more effective branching strategy and stronger variable fixing procedures. The second one is the integration of a dedicated heuristic in the exact approach. The experimental results show a strong average reduction of the computational time and the number of branching nodes. This also mitigates the anticipated termination of the algorithm due to the exhaustion of the memory on the largest benchmark instances.

## 1 INTRODUCTION


The *Weighted Safe Set Problem* (WSSP) is the problem of finding a *safe set* of minimum weight on a connected undirected graph  $G = (V, E)$  with weights  $w : V \rightarrow \mathbb{R}^+$  defined over the vertices. To define a safe set it is necessary to introduce some concepts. The *weight*  $w(C)$  of a subset of vertices  $C \subseteq V$  is the sum of the weights of all vertices in it. Two subsets  $C_1, C_2 \subseteq V$  are considered *adjacent* (and we write  $C_1 \bowtie C_2$ ) if there is at least one edge  $(u, v)$  in the graph such that  $u \in C_1$  and  $v \in C_2$ . Given a set  $S \subseteq V$  of vertices and its complementary set  $U = V \setminus S$ ,  $C_G(S)$  is the collection of *all maximal connected components* induced by  $S$  in graph  $G$  and  $C_G(U)$  the corresponding collection induced by  $U$ . If the weight of each component of  $C_G(S)$  is not smaller than that of each adjacent component of  $C_G(U)$ , we denote  $S$  as a *safe set*, the components in  $C_G(S)$  as *safe components* and those in  $C_G(U)$  as *unsafe components*. Given a safe component  $S_c$  and an adjacent unsafe component  $U_c$ , we will refer to the condition that  $w(S_c) \geq w(U_c)$  as *safety constraint*.


The WSSP was first introduced in its unweighted version (the weight of every vertex is 1) by Fujita et al.

(2014), while Bapat et al. (2016) defined the general weighted version. The real-world applications proposed in these articles concern the choice of buildings or areas to be converted into temporary shelters in order to host people from the adjacent dangerous areas, and the identification of a critical community which can control the totality of a social network.

The problem is NP-hard for both unweighted and weighted versions (Fujita et al., 2016; Bapat et al., 2016). Its parameterised complexity has been studied by Àgueda et al. (2018) and Belmonte et al. (2020). Exact algorithms for specific instances have been proposed by Fujita et al. (2016), Àgueda et al. (2018) and Cordone and Franchi (2023). Approximation algorithms have been proposed by Bapat et al. (2018) and Ehard and Rautenbach (2020). Heuristic approaches have been presented by Macambira et al. (2019) and Boggio Tomasaz et al. (2023b). Finally, exact (non-polynomial) approaches have been presented by Macambira et al. (2019), Hosteins (2020), Malaguti and Pedrotti (2023) and Boggio Tomasaz et al. (2023a). The last one solves instances up to 60 vertices in a time limit of one hour.

In this article we revisit the combinatorial branch and bound algorithm presented in Boggio Tomasaz et al. (2023a), introducing a relaxation that is theoretically weaker, but most of the time equivalent, and allows more efficient branching rules and reduction

<sup>a</sup>  <https://orcid.org/0000-0003-3334-6238>

<sup>b</sup>  <https://orcid.org/0000-0002-5439-1743>

procedures leading to a simpler and faster algorithm. We also discuss the contribution provided by a good heuristic initialisation.

## 2 THE REVISITED ALGORITHM

Boggio Tomasaz et al. (2023a) present a combinatorial branch and bound algorithm for the *WSSP* with the following main features. The algorithm builds a binary branching tree. It progressively assigns the vertices either to the safe set  $S$  or the unsafe one,  $U$ , thus extending the current safe and unsafe components or creating new ones. The branching tree is explored with the best-bound strategy, that is, always visiting the open node with the minimum lower bound. In fact, in any branching node, the algorithm computes a lower bound through a sequence of relaxations that combine manipulations of the graph and of an Integer Linear Programming formulation. Moreover, it applies logical reduction procedures to move free (i.e. unassigned) vertices into  $S$  or  $U$ . The computation of the lower bound requires that at most one of the connected components of the current safe set  $S$  have free adjacent vertices. The other safe components must be completely surrounded by unsafe vertices. This is a basic drawback of the approach for two reasons. First, it limits the branching rule, since not all vertices can be selected and assigned to  $S$  and  $U$  in the two children nodes. Second, it hampers the logical reduction procedures, since they cannot be applied when the result creates several safe components with adjacent free vertices, violating the requirement. On sufficiently sparse instances (with a threshold experimentally identified between 0.1 and 0.2), these limitations are so strong that it actually proves more effective to resort to a trivial lower bound (the weight of the current safe set, or of the maximum weight of a current unsafe component) in order to preserve the remaining parts of the algorithm.

In each branching node, the algorithm also generates a heuristic solution compatible with the branching constraints. While these solutions are computed very quickly, their quality is bad. This is another drawback of the approach. On small instances, in fact, the branching process is still fast enough to find a provably optimal solution in short time. On large instances, on the contrary, the time limit sometimes expires when the best known solution is still far from the optimum. Moreover, a bad upper bound implies an increase in the number of open nodes during the visit of the branching tree, possibly leading to exhaust the available memory.

In this paper we revise the branch and bound al-

gorithm in order to alleviate its two main drawbacks. To address the first issue, we introduce an additional graph manipulation that in principle yields a further relaxed problem, and therefore a weaker lower bound. The empirical results, however, show that the downside is most of the time null or negligible, while the possibility to select the branching vertex more freely and to apply all reduction procedures gives a beneficial contribution. Overall, this allows to simplify the algorithm, avoiding the distinction between sparse and dense instances, and to exploit the strengths of all components of the algorithm at the same time. To address the second issue, we apply a refined heuristic before the branch and bound algorithm to obtain a starting upper bound.

In the following section, we present the relaxation proposed by Boggio Tomasaz et al. (2023a) and the additional manipulation here introduced, discussing its potential effects and the reasons that often make them irrelevant. Section 2.2 describes the branching rule and logical reduction procedures, focusing on the more general applicability guaranteed by the new relaxation. Section 2.3 deals with the heuristic used to initialise the search.

### 2.1 A Weaker but More Versatile Relaxation

Any subproblem in the branching tree is characterised by a partition  $(S, U, F)$  of the vertex set  $V$  into the set  $S$  of fixed safe vertices, the set  $U$  of fixed unsafe vertices and the set  $F = V \setminus (S \cup U)$  of free vertices, not already fixed. We also define the subset of free vertices adjacent to both  $S$  and  $U$ :

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

In this work we present and prove the following new proposition, that allows to manipulate graph  $G$  in order to obtain a relaxation of the given *WSSP*.

**Proposition 1.** *Adding to graph  $G = (V, E)$  an edge  $(u, v)$  with  $u, v \in S$  provides a relaxation of subproblem  $(S, U, F)$ .*

*Proof.* We show that in the modified graph all feasible solutions keep their value and remain feasible. Consider two fixed safe vertices  $u, v \in S$  and a solution  $\tilde{S}$  for subproblem  $(S, U, F)$ . Two cases are possible:

1.  $u$  and  $v$  are in the same safe component in  $\tilde{S}$ : in such a situation adding edge  $(u, v)$  does not change the components and the value of  $\tilde{S}$ ;
2.  $u$  and  $v$  are in different safe components in  $\tilde{S}$ : by adding  $(u, v)$ , the two safe components merge, while all unsafe components and the value of the objective remain unchanged.

□

Applying this proposition to all pairs of safe vertices merges them into a single safe component. The following propositions, introduced and proved in Boggio Tomasaz et al. (2023a), will be combined with the previous one to further relax the problem.

**Proposition 2.** *Removing from graph  $G = (V, E)$  an edge  $(u, v)$  with  $u \in U$  and  $v \in F$  or both  $u, v \in F'$  provides a relaxation of subproblem  $(S, U, F)$ .*

**Proposition 3.** *Replacing in graph  $G = (V, E)$  an edge  $(u, v)$  with an edge  $(s, v)$  provides a relaxation of subproblem  $(S, U, F)$  if  $s \in S$ ,  $u \in F'$ ,  $v \in F \setminus F'$  and  $(s, u) \in E$ .*

Notice that Proposition 3 has been slightly rephrased in an equivalent form in order to adapt it to the new context.

The three propositions above allow to manipulate the graph 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 resulting manipulated graph, which we call  $G_m = (V, E_m)$ , consists of: i) a clique  $S$  of fixed safe vertices; ii) a collection of unsafe components  $C_{G_m}(U)$  with the original internal edges; iii) an independent set  $F'$  of vertices, which are linked to  $S$  by at least one edge, to  $U$  by exactly one edge, to  $F$  by no edge; iv) the remaining set of free vertices  $F \setminus F'$  with the original internal edges, disconnected from  $F'$  and  $U$ , but possibly connected with  $S$ . As a consequence, the unique safe component can be enlarged including vertices in  $F$ , whereas each unsafe component  $U_l \in C_{G_m}(U)$  can be enlarged including only the vertices in  $F'$  that are directly adjacent to  $U_l$ . These vertices form set:

$$F'_l := \{v \in F' \mid \exists s \in S : (s, v) \in E_m \wedge \exists u \in U_l : (u, v) \in E_m\}$$

Given the current subproblem  $(S, U, F)$  and the manipulated graph  $G_m$ , we can introduce the following Integer Linear Programming formulation. Let  $x_v = 1$  when vertex  $v$  belongs to the solution, and  $x_v = 0$  otherwise. The auxiliary variable  $\sigma$  represents the weight of the free vertices assigned to the unique safe component, while  $\tau_l$  represents the weight of the free vertices assigned to the unsafe component  $U_l$ .

$$\min w(S) + \sigma \quad (1a)$$

subject to:

$$w(S) + \sigma \geq w(U_l) + \tau_l \quad U_l \in C_{G_m}(U) \quad (1b)$$

$$\sigma = \sum_{v \in F} w_v \cdot x_v \quad (1c)$$

$$\tau_l = \sum_{v \in F'_l} w_v \cdot (1 - x_v) \quad U_l \in C_{G_m}(U) \quad (1d)$$

$$\sigma, \tau_l \geq 0 \quad U_l \in C_{G_m}(U) \quad (1e)$$

$$x_v \in \{0, 1\} \quad v \in F \quad (1f)$$

where objective (1a) is the weight of the safe set and constraints (1b) are the safety conditions between the unique safe component and the currently known unsafe ones. Constraints (1c) and (1d) define the auxiliary variables and constraints (1e) and (1f) the domains of all variables. Since the safety constraints (1b) neglect the components that the free vertices could still form, Formulation (1) is a relaxation of the subproblem  $(S, U, F)$ .

Now, we replace constraints (1c), (1d) and (1f) with:

$$\tau_l \leq w(F'_l) \quad \forall U_l \in C_{G_m}(U) \quad (2)$$

$$\sigma + \sum_{U_l \in C_{G_m}(U)} \tau_l \geq w(F') \quad (3)$$

The former derives from majorising  $(1 - x_v)$  with 1 in (1d), the latter from summing constraints (1c) and (1d). This yields a relaxation of Formulation (1):

$$\min w(S) + \sigma \quad (4a)$$

subject to:

$$w(S) + \sigma \geq w(U_l) + \tau_l \quad U_l \in C_{G_m}(U) \quad (4b)$$

$$\sigma + \sum_{U_l \in C_{G_m}(U)} \tau_l \geq w(F') \quad (4c)$$

$$\sigma \geq 0 \quad (4d)$$

$$0 \leq \tau_l \leq w(F'_l) \quad U_l \in C_{G_m}(U) \quad (4e)$$

which we solve with a simple readaptation of the algorithm proposed in Boggio Tomasaz et al. (2023a).

With respect to the original relaxation, the new approach provides a weaker bound, due to the fact that forcing all safe vertices in the same safe component generates new feasible solutions, that could improve the value of the optimum. However, the empirical results show that the branching nodes in which the number of safe components is larger than one are very rare, and that all benchmark instances have a connected optimal solution. This can be explained in view of the properties of random graphs, which asymptotically tend to become connected as their number of vertices increases, as discussed in Cordone and Franchi (2023).

## 2.2 Reduction Procedures and Branching Rule

Given a subproblem  $(S, U, F)$ , suitable reduction procedures allow to assign some of the free vertices to the safe or the unsafe set without affecting the value of the optimum. The following propositions, proved in Boggio Tomasaz et al. (2023a), guarantee the correctness of these procedures.

**Proposition 4.** *Given a subproblem  $(S, U, F)$ , if a component  $F_l \in C_G(F)$  is not adjacent to  $S$  and has weight  $w(F_l)$  smaller than that of an adjacent unsafe component, all its vertices can be assigned to  $U$ .*

This proposition also allows to test the existence of feasible solutions. In fact, after its application, subproblem  $(S, U, F)$  is feasible if and only if  $S \cup F$  is feasible. The best solution thus generated yields the upper bound in the original branch and bound approach.

The following proposition extends the test with an implicit branching operation. We remind that the sign  $\bowtie$  represents the *adjacency* relation between subsets of vertices.

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

$$w(C) - w_f < \sum_{\substack{U_j \in C_G(U): \\ U_j \bowtie \{f\}}} w(U_j) + w_f \quad (5)$$

*then vertex  $f$  can be assigned to  $S$ .*

Finally, another implicit branching operation fixes vertices into the safe or the unsafe set, based on a comparison with the currently best known value.

**Proposition 6.** *Let  $(S, U, F)$  be a subproblem,  $f \in F$  a free vertex and  $\bar{S}$  the best known feasible solution. If*

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

*then vertex  $f$  can be assigned to  $U$ . If*

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

*then vertex  $f$  can be assigned to  $S$ .*

Notice that moving free vertices into  $S$  can create new safe components. This was forbidden in the original algorithm, thus curtailing its performance. On the contrary, the new relaxation is compatible with all the operations listed above. Since they reduce the size of the problem and this can eventually increase the lower bound, adopting the weaker relaxation is not necessarily disadvantageous.

Similarly, the branching mechanism of the original approach is simplified and extended by the new

relaxation. The basic idea of the branching rule is to select the free vertex of maximum weight (and maximum degree as a tie breaker), since moving it into either  $S$  or  $U$  is very likely to increase the lower bound by the maximum amount. However, such a choice in general creates new safe components, which was previously forbidden. The branching rule was therefore restricted to select the free vertex of maximum weight adjacent to  $S$  or (in case none exists) the free vertex of maximum weight adjacent to  $U$ . In sparse instances, this restriction was particularly limiting. By contrast, the new relaxation allows to apply the more effective basic selection rule.

## 2.3 Heuristic Procedure

Proposition 4 allows to compute feasible solutions during the exploration of the branching tree. However, the quality of these solutions can be very poor, especially at the beginning, when the number of free vertices is high. To overcome this issue, we can feed the algorithm with an initial upper bound generated by a heuristic procedure in order to amplify the effect of the pruning.

The state-of-the-art heuristics for the WSSP are the ones presented by Boggio Tomasaz et al. (2023b). The most promising one is the *Simple Delayed Termination (SDT)*. This can be explained as a sort of *Large Neighbourhood Search* which first builds a feasible solution and then alternatively enlarges it with redundant vertices and reduces it to a minimal feasible subset.

The algorithm starts with an unfeasible empty set  $S = \emptyset$  and iteratively extends it with vertices in  $V \setminus S$ . The choice of the vertex to insert is randomised exploiting a *Restricted Candidate List (RCL)* mechanism that favours the vertices with many adjacent vertices in  $V \setminus S$ . Once a feasible solution is found, instead of stopping, the algorithm further enlarges  $S$  *delaying its termination* for a certain number of iterations, proportional to the size of the instance. In this phase, the insertion deterministically selects the vertex of minimum weight (and maximum degree to break ties) adjacent to  $S$ . Moreover, a destructive procedure is applied to every feasible solution  $S$  found, in order to compute a minimal feasible solution  $S' \subseteq S$ . Starting from  $S' = S$ , the destructive heuristic considers each vertex and tries to erase it. If the resulting set is unfeasible, the vertex is reinserted in  $S'$ ; otherwise, it remains discarded. The vertices are processed in non-increasing order of weight (and increasing order of degree as a tie breaker). The computational complexity of the SDT heuristic is  $O(|V|^2(|V| + |E|))$  (Boggio Tomasaz et al., 2023b).



Since the initial phase is randomised, the heuristic is restarted several times and returns the best feasible solution found among all the attempts.

### 3 COMPUTATIONAL RESULTS

In this section we report the results obtained by the computational experiments conducted on the revisited branch and bound algorithm, that in the following we will denote as RevBB. All the experiments are conducted on a Linux server, with processor Intel Xeon E5-2620 2.1 GHz and 16 GB of RAM, that is the same machine used in Boggio Tomasaz et al. (2023a). The algorithm is coded in C99 and compiled with GNU GCC 8.3.0, and runs in a single thread.

The instances are the ones presented in the literature by Macambira et al. (2019) and the ones added by Hosteins (2020) and extended in Boggio Tomasaz et al. (2023a). They can be found at this link: <https://homes.di.unimi.it/cordone/research/wssp.html>. The graphs are randomly generated in accordance with the Erdős-Renyi model. In the former set the number of vertices progressively rises from 10 to 30 and the number of edges is  $|E| = \delta \cdot |V| \cdot (|V| - 1) / 2$ , where the *density* parameter assumes values  $\delta \in \{0.3, 0.5, 0.7\}$ . There are weighted instances with random weights extracted from a uniform distribution in  $\{1, \dots, 100\}$  and unweighted instances with weights uniformly set to 1. This benchmark consists of  $21 \cdot 3 \cdot 2 = 126$  instances. In the latter set, the number of vertices is  $|V| \in \{20, 25, 30, 35, 40, 50, 60\}$  and the *density* parameter assumes values  $\delta \in \{0.1, 0.2, 0.3, 0.4\}$ . For each combination of  $|V|$  and  $\delta$  there are 5 weighted instances with random uniform integer weights between 1 and 10 and 5 unweighted instances, that is  $7 \cdot 4 \cdot 5 \cdot 2 = 280$  instances overall.

#### 3.1 Effect of the Revised Relaxation

In this section we compare the computational results of the revisited algorithm with those reported in Boggio Tomasaz et al. (2023a) for the original branch and bound within the time limit of one hour. Tables 1 and 2 provide in the first two columns the density  $\delta$  and the size  $|V|$  of the instances. The following four columns show the gap (formatted as a percentage), the number of solved instances in each group, the computational time in seconds (CPU) and the number of branching nodes (BN). Each value is averaged over the 5 instances with the density and size indicated in the corresponding row. The gap is computed as  $(UB - LB) / LB$ , where  $UB$  and  $LB$  are the best upper and lower bounds found by the algorithm. The last

four columns contain the same information for algorithm RevBB. The computational times and the numbers of branching nodes marked with \* point out that the exhaustion of memory anticipated the termination on some instances. The symbol OM indicates that all 5 instances of a group went “out of memory”.

In the weighted case (see Table 1), the revisited algorithm leaves only 2 unsolved instances out of 140, instead of 5. One of the newly solved instances was previously terminated because it exceeded the available memory. The two remaining unsolved instances strictly improve both the upper and the lower bound: the residual gap is about one third of its original value. The computational time and the number of branching nodes is roughly halved, except for the instances in which the memory is no longer exhausted and the computation can proceed for a longer time.

Considering the unweighted instances (see Table 2), the unsolved ones decrease from 21 to 16 out of 140. The ones with an out-of-memory termination decrease from 11 to 10. On average, the gaps of the 16 unsolved instances are more than 2 times smaller, with the exception of the instances with  $\delta = 0.2$ , for which there is a moderate increase. In this specific case, the weaker lower bound is not completely counterbalanced by its advantages, leading to an overconsumption of memory. The comparison between the computational times and the number of branching nodes still shows an improvement, in particular for the denser instances, but less pronounced than in the weighted case. The instances with  $|V| = 60$  and  $\delta \in \{0.1, 0.2\}$  are difficult to compare from this point of view because of the anticipated termination.

#### 3.2 Effect of the Initial Heuristic

In this section we present the results obtained by exploiting the SDT heuristic presented in Boggio Tomasaz et al. (2023b) as an initial incumbent for the RevBB algorithm. Tables 3 and 4 share the same structure of Tables 1 and 2, but show the results obtained by RevBB and SDT+RevBB. Since the RevBB algorithm solves to optimality the instances with  $|V| \leq 35$  in few seconds, we do not consider those instances in the comparison. The time limit for SDT+RevBB is one hour, to obtain a fair comparison with the original branch and bound. The computational time for the SDT heuristic is set to 0.1 seconds for  $|V| = 40$ , 1 second for  $|V| = 50$  and 10 seconds for  $|V| = 60$ . The aim is to keep into account the longer time required by each iteration as the size of the instance grows, but also to slightly increase the number of iterations, that is in the range of a few thousands. The heuristic runs with the same parame-

Table 1: Comparison between the algorithm in Boggio Tomasaz et al. (2023a) and RevBB over all weighted instances of the benchmark.

$\delta$	$ V $	Boggio Tomasaz et al. (2023a)				RevBB			
		gap	solved	CPU	BN	gap	solved	CPU	BN
0.1	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
	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
0.2	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
	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
0.3	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
	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
0.4	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
	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

ter tuning suggested by the experiments described in Boggio Tomasaz et al. (2023b).

With respect to the weighted problem (see Table 3), the number of unsolved instances remains 2 out of 60, but one instance improves both the lower and the upper bound. The improvements on the computational time and the number of branching nodes are less marked and uniform than in Table 1. In general, they decrease, but the denser instances with  $|V| = 40$  and  $|V| = 60$  show a moderate increase.

As for the unweighted problem (see Table 4), the number of unsolved instances decreases from 16 to 15. The average gaps sharply decrease on the sparse instances, where both the lower and the upper bounds improve. For  $\delta \in \{0.1, 0.2\}$ , the computational times and the branching nodes significantly decrease. For  $\delta = 0.3$ , however, they decrease slightly, whereas for  $\delta = 0.4$  they undergo a moderate increase, as in the weighted case. In short, the heuristic initialisation gives a useful contribution mainly on the sparser instances, that are the harder ones for the exact algo-

rithm.

### 3.3 Comparison with the State of the Art

Currently, the best performing exact algorithm for the WSSP is the combinatorial branch and bound by Boggio Tomasaz et al. (2023a), except for the sparsest instances ( $\delta = 0.3$ ) introduced by Macambira et al. (2019), on which the branch and cut by Malaguti and Pedrotti (2023) requires lower computational times. The machine used by the latter algorithm is an Intel i7-4790 processor running at 3.60 GHz and endowed with 32 GB RAM. It is roughly comparable, but probably faster than the one used by the branch and bound. Table 5 reports the running times required by the two algorithms to solve to optimality each instance of this benchmark, rounded to the first decimal digit, as in Malaguti and Pedrotti (2023). The values for  $|V| = 10$  are missing in the original reference. Label MP denotes the branch and cut algorithm,

Table 2: Comparison between the algorithm in Boggio Tomasaz et al. (2023a) and RevBB over all unweighted instances of the benchmark.

$\delta$	$ V $	Boggio Tomasaz et al. (2023a)				RevBB			
		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	1716.793*	OM	1.25%	4	2165.374	152910911
	60	142.42%	0	1258.089*	OM	68.40%	0	2193.231*	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	2024.438*	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: Comparison between the RevBB algorithm without and with an initial heuristic solution over all the weighted instances of the benchmark.

$\delta$	$ V $	RevBB				SDT+RevBB			
		gap	solved	CPU	BN	gap	solved	CPU	BN
0.1	40	-	5	2.788	299276	-	5	2.549	258594
	50	-	5	111.651	8102922	-	5	99.890	7073450
	60	-	5	1199.706	62877972	-	5	874.839	43777463
0.2	40	-	5	3.997	277123	-	5	3.421	222507
	50	-	5	307.728	14711729	-	5	249.368	11734634
	60	1.82%	3	2290.738	74773767	1.07%	3	2160.774	68785511
0.3	40	-	5	4.898	266300	-	5	4.524	235375
	50	-	5	47.728	1665147	-	5	45.325	1541978
	60	-	5	743.674	17805962	-	5	694.451	16086338
0.4	40	-	5	2.059	89991	-	5	2.272	94113
	50	-	5	27.720	773785	-	5	24.890	676289
	60	-	5	271.408	5005143	-	5	320.006	5558855

SDT+RevBB the revised branch and bound initialised with a very short run of the SDT heuristic (0.001 seconds). Many of the reported times are too short to

allow a meaningful comparison (0.0 stands for values  $< 0.05$ ). However, on the denser instances the combinatorial approach confirms to be order of mag-

Table 4: Comparison between the RevBB algorithm without and with an initial heuristic solution over all the unweighted instances of the benchmark.

$\delta$	$ V $	RevBB				SDT+RevBB			
		gap	solved	CPU	BN	gap	solved	CPU	BN
0.1	40	-	5	44.817	4778665	-	5	25.922	2718502
	50	1.25%	4	2165.374	152910911	-	5	1342.176	100049161
	60	68.40%	0	2193.231	OM	14.40%	0	3032.005	186290447*
0.2	40	-	5	47.749	3390965	-	5	24.351	1804663
	50	-	5	1974.602	100802247	-	5	1234.439	65906293
	60	33.90%	0	2024.438	OM	10.40%	0	3600.000	129482857
0.3	40	-	5	14.626	837818	-	5	13.193	786002
	50	-	5	480.459	18963773	-	5	442.337	18425442
	60	4.29%	0	3600.000	94802520	4.29%	0	3600.000	94782363
0.4	40	-	5	5.800	270571	-	5	5.236	249147
	50	-	5	56.295	1689201	-	5	59.519	1827820
	60	-	5	710.371	13912611	-	5	900.183	17445371

Table 5: Comparison between the branch and cut of Malaguti and Pedrotti (2023) and the RevBB algorithm initialised by the SDT heuristic on the benchmark by Macambira et al. (2019).

$ V $	Weighted						Unweighted					
	$\delta = 0.3$		$\delta = 0.5$		$\delta = 0.7$		$\delta = 0.3$		$\delta = 0.5$		$\delta = 0.7$	
	MP	SDT+ RevBB	MP	SDT+ RevBB	MP	SDT+ RevBB	MP	SDT+ RevBB	MP	SDT+ RevBB	MP	SDT+ RevBB
10		0.0		0.0		0.0		0.0		0.0		0.0
11	0.0	0.0	0.1	0.0	1.4	0.0	0.0	0.0	0.2	0.0	19.6	0.0
12	0.0	0.0	0.1	0.0	7.3	0.0	0.0	0.0	0.1	0.0	4.3	0.0
13	0.0	0.0	0.0	0.0	1.6	0.0	0.0	0.0	0.0	0.0	27.5	0.0
14	0.0	0.0	0.1	0.0	2.7	0.0	0.0	0.0	0.2	0.0	38.8	0.0
15	0.0	0.0	0.1	0.0	2.6	0.0	0.0	0.0	0.3	0.0	10.1	0.0
16	0.0	0.0	0.2	0.0	7.2	0.0	0.0	0.0	0.8	0.0	33.1	0.0
17	0.0	0.0	0.1	0.0	4.3	0.0	0.0	0.0	1.5	0.0	455.9	0.0
18	0.0	0.0	0.5	0.0	2.8	0.0	0.0	0.0	2.6	0.0	80.7	0.0
19	0.0	0.0	0.4	0.0	5.8	0.0	0.0	0.0	0.5	0.0	24.9	0.0
20	0.0	0.0	0.2	0.0	25.4	0.0	0.0	0.0	0.8	0.0	93.3	0.0
21	0.0	0.0	0.5	0.0	9.6	0.0	0.0	0.0	0.7	0.0	140.0	0.0
22	0.0	0.0	0.1	0.0	1.9	0.0	0.0	0.0	1.0	0.0	52.1	0.0
23	0.1	0.0	0.7	0.0	12.0	0.0	0.0	0.0	3.7	0.0	418.1	0.0
24	0.0	0.0	1.6	0.0	7.2	0.0	0.0	0.0	0.2	0.1	311.0	0.0
25	0.1	0.0	1.2	0.0	13.3	0.0	0.1	0.1	0.5	0.1	80.7	0.1
26	0.1	0.1	1.4	0.1	58.4	0.0	0.1	0.1	6.5	0.1	878.8	0.0
27	0.1	0.1	1.6	0.1	26.6	0.1	0.0	0.1	3.6	0.1	2416.3	0.1
28	0.0	0.0	0.4	0.1	14.7	0.0	0.1	0.2	3.6	0.2	111.2	0.0
29	0.0	0.2	0.8	0.3	65.9	0.2	0.1	0.2	11.1	0.2	2418.2	0.5
30	0.2	0.2	0.1	0.3	46.8	0.3	0.1	0.5	11.1	0.4	439.1	0.0

nitide faster than the branch and cut. On the sparser instances, the two approaches are now equivalent for the weighted instances, while the branch and cut approach is still faster on the unweighted ones.

Table 6 reports the results for the benchmark introduced by Hosteins (2020). For the branch and cut algorithm, it provides the average execution time in seconds, the average residual gap and the number of

solved instances after one hour of computation. Since the combinatorial branch and bound solves all instances within one hour, we report only the average computational time. The SDT heuristic for the initialisation phase runs for 0.001 seconds on the instances up to 35 vertices, 0.1 seconds for  $|V| = 40$ , 1 second for  $|V| = 50$  and 10 seconds for  $|V| = 60$ . The rationale is that the smaller instances could be solved



Table 6: Comparison between the branch and cut of Malaguti and Pedrotti (2023) and the RevBB algorithm initialised by the SDT heuristic on the benchmark by Hosteins (2020).

V	$\delta$	Weighted			SDT+RevBB CPU	Unweighted			SDT+RevBB CPU
		MP CPU	gap	solved		MP CPU	gap	solved	
20	0.1	0.8	-	5	0.0	1.3	-	5	0.0
25	0.1	4.9	-	5	0.0	5.2	-	5	0.0
30	0.1	20.1	-	5	0.0	32.4	-	5	0.1
35	0.1	55.0	-	5	0.1	469.4	-	5	1.8
40	0.1	1 244.2	2.2%	4	2.5	2 976.1	19.0%	1	25.9
50	0.1	3 600.0	22.7%	0	99.9	3 600.0	37.6%	0	1 342.2
20	0.2	0.3	-	5	0.0	0.6	-	5	0.0
25	0.2	1.9	-	5	0.0	4.9	-	5	0.1
30	0.2	10.6	-	5	0.1	45.2	-	5	0.4
35	0.2	195.3	-	5	1.2	2 825.1	7.7%	2	7.1
40	0.2	1 207.8	-	5	3.4	3 116.9	29.2%	0	24.4
50	0.2	3 600.0	33.1%	0	249.4	3 600.0	45.8%	0	1 234.4
20	0.3	0.2	-	5	0.0	0.2	-	5	0.0
25	0.3	2.3	-	5	0.0	12.5	-	5	0.1
30	0.3	22.5	-	5	0.2	174.3	-	5	0.4
35	0.3	326.6	-	5	1.0	3 141.5	10.5%	1	4.2
40	0.3	2 400.3	2.1%	3	4.5	3 368.4	30.3%	0	13.2
50	0.3	3 600.0	24.1%	0	45.3	3 600.0	43.2%	0	442.3
20	0.4	0.5	-	5	0.0	1.1	-	5	0.0
25	0.4	5.1	-	5	0.0	40.8	-	5	0.1
30	0.4	34.2	-	5	0.2	256.6	-	5	0.4
35	0.4	404.6	-	5	1.2	3 486.8	9.8%	1	2.9
40	0.4	1 367.2	-	5	2.3	3 600.0	26.6%	0	5.2
50	0.4	3 600.0	17.8%	0	24.9	3 600.0	39.6%	0	59.5

very quickly even without an initialisation, whereas the time for the larger ones is tuned so as to slightly increase the number of iterations of the SDT heuristic as the size of the instance grows. On this benchmark the original algorithm was already faster than the branch and cut for all classes of instances. As discussed in the previous section, the revised algorithm further reduces the time in most cases. Even when the time increases, however, the comparison with the branch and cut remains favourable to the combinatorial approach.

#### 4 CONCLUSIONS

In this article we presented a revised branch and bound algorithm for the *Weighted Safe Set Problem*. Its main strength lies in a theoretically weaker, but more versatile, relaxation, which allows to exploit a more efficient branching rule and reductions. The new algorithm outperforms the previous one on the vast majority of instances in the literature, solving to optimality instances with up to 60 vertices. The use of an existing heuristic to initialise the search is rather

useful on the largest instances (60 vertices), but hardly necessary on the smaller ones (up to 35 vertices). On denser instances, however, the additional effort associated to the initialisation is not always justified. A finer tuning of this aspect, therefore, seems a promising development.

#### REFERENCES

Águeda, R., Cohen, N., Fujita, S., Legay, S., Manoussakis, Y., Matsui, Y., Montero, L., Naserasr, R., Ono, H., Otachi, Y., Sakuma, T., Tuza, Z., and Xu, R. (2018). Safe sets in graphs: Graph classes and structural parameters. *Journal of Combinatorial Optimization*, 36(4):1221–1242.

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

Bapat, R. B., Fujita, S., Legay, S., Manoussakis, Y., Matsui, Y., Sakuma, T., and Tuza, Z. (2018). Safe sets, network majority on weighted trees. *Networks*, 71(1):81–92.

Belmonte, R., Hanaka, T., Katsikarelis, I., Lampis, M., Ono, H., and Otachi, Y. (2020). Parameterized com-

- plexity of safe set. *Journal of Graph Algorithms and Applications*, 24(3):215—245.
- Boggio Tomasaz, A., Cordone, R., and Hosteins, P. (2023a). A combinatorial branch and bound for the safe set problem. *Networks*, 81(4):445–464.
- Boggio Tomasaz, A., Cordone, R., and Hosteins, P. (2023b). Constructive–destructive heuristics for the safe set problem. *Computers & Operations Research*, 159:106311.
- Cordone, R. and 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.
- Ehard, S. and Rautenbach, D. (2020). Approximating connected safe sets in weighted trees. *Discrete Applied Mathematics*, 281:216–223.
- Fujita, S., MacGillivray, G., and Sakuma, T. (2014). Bordeaux graph workshop: Safe set problem on graphs. <https://bgw.labri.fr/2014/bgw2014-booklet.pdf>.
- Fujita, S., MacGillivray, G., and Sakuma, T. (2016). Safe set problem on graphs. *Discrete Applied Mathematics*, 215:106–111.
- Hosteins, P. (2020). A compact mixed integer linear formulation for safe set problems. *Optimization Letters*, 14(8):2127–2148.
- Macambira, A. F. U., Simonetti, L., Barbalho, H., Silva, P. H. G., and Maculan, N. (2019). A new formulation for the safe set problem on graphs. *Computers and Operations Research*, 111:346–356.
- Malaguti, E. and Pedrotti, V. (2023). Models and algorithms for the weighted safe set problem. *Discrete Applied Mathematics*, 329:23–34.