Constructive–destructive heuristics for the Safe Set Problem

Constructive-destructive heuristics for the Safe Set Problem

Alberto Boggio Tomasaz*

Università degli Studi di Milano, Dipartimento di Informatica

Roberto Cordone

Università degli Studi di Milano, Dipartimento di Informatica

Pierre Hosteins

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

Abstract

The *Weighted Safe Set Problem* aims to determine in an undirected graph a subset of vertices such that the weights of the connected components they induce exceed the weights of the adjacent components induced by the complementary subset. We tackle the problem with various approaches based on randomised constructive and destructive procedures, comparing them with each other and with the only other existing heuristic approach, that is a randomised destructive heuristic. The experiments concern all the available benchmark instances (random graphs up to 60 vertices), but also new benchmark instances with larger size and different topologies, namely random, small-world, regular, planar, grid and real-world graphs up to 300 vertices. Dense random graphs, in fact, appear to become progressively easier to solve as their size grows. On the other instances, the best performance is obtained by combining a randomised constructive procedure, a deterministic destructive procedure, and delayed termination conditions that allow a deeper exploration of the feasible region.

Keywords: Graph partitioning, Safe Set Problem, Greedy Randomized Adaptive Search Procedure, Large Neighbourhood Search

1. Introduction

The problems posed by the safety of networks and by the interplay between the subnetworks that compose them has recently been the object of a strong interest, with a specific focus on communities in social networks. This has given impulse to the proposal of graph optimisation models that could capture the basic features of such problems. One of these models is the *Weighted Safe Set Problem* (Bapat et al., 2016), that can be roughly defined as the search for a minimal collection of subnetworks whose control allows to indirectly dominate the whole network. More formally, a connected undirected graph $G = (V, E)$ has a positive weight

*Corresponding author

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

function $w : V \to \mathbb{Q}_+$ defined on its vertices. Any subset of vertices $S \subseteq V$ induces on G a collection $\mathcal{C}_G(S)$ of maximal connected components, while its complement $V \setminus S$ induces a second collection $\mathcal{C}_G(V \setminus S)$ of components that form the remaining part of the graph. We denote as *safe set* a nonempty subset $S \subseteq V$ such that each connected component $S_i \in \mathcal{C}_G(S)$ has a weight not smaller than the weight of any adjacent connected component $U_j \in \mathcal{C}_G(V \setminus S)$. The weight of a subset is trivially the sum of the weights of its elements ($w(T) = \sum_{v \in T} w_v$). The adjacency relation between disjoint subsets extends the corresponding relation between vertices: S_i and U_j are adjacent ($S_i \bowtie U_j$) when there exists an edge between a vertex of S_i and a vertex of U_j. The *Weighted Safe Set Problem* (*WSSP*) is the search for a safe set of minimum weight and it can be expressed with the following combinatorial formulation:

$$\min w(S) \tag{1}$$

$$\text{s.t.} \quad S \subseteq V \tag{2}$$

$$|S| \geq 1 \tag{3}$$

$$w(S_i) \geq w(U_j) \qquad S_i \in \mathcal{C}_G(S), U_j \in \mathcal{C}_G(V \setminus S), S_i \bowtie U_j \tag{4}$$

where Constraint (2) states that the solution is a subset of vertices, Constraint (3) that it is nonempty and Constraints (4) impose the safety conditions.

The literature proposes several possible real-world applications for the *WSSP*. Bapat et al. (2016) use it to model the search for a leading subset of a physical or virtual community represented by a network, where the weights of the nodes measure their capacity to influence other nodes. Fujita et al. (2016) apply the same model to the location of emergency refuges inside a building, where the nodes correspond to rooms and the aim is to offer shelter for the people in the whole building using the minimum possible space.

Motivated by the fact that in the literature there exists only a single heuristic approach to this problem (Macambira et al., 2019), this paper introduces a number of new metaheuristic approaches to the *WSSP*. The first two approaches follow the framework of the *Greedy Randomised Adaptive Search Procedure* (*GRASP*) (Feo and Resende, 1989). They are based on a randomised constructive mechanism that allows multiple restarts, followed by a destructive post-processing phase to improve the result. Other three approaches improve the exploration of the solution space by extending the constructive process after the first feasible solution has been found, to visit several other ones in its surroundings. As they alternate between generating redundant solutions and reducing them by removing suitable vertices, these approaches show strong similarities to the *Large Neighbourhood Search* (*LNS*) framework (Shaw, 1998). Two of the latter algorithms prove particularly effective and outperform the randomised destructive metaheuristic proposed in the literature (Macambira et al., 2019). They also improve the best known results provided by the state-of-the-art exact algorithm on the available benchmark instances for which the optimal solution is currently unknown.

The available benchmark instances of the *WSSP* are random graphs of small size, up to 60 vertices, that have been proposed to test exact algorithms. We introduce new much larger instances, up to 300 vertices, with

the same structure. Our experience, however, shows that random graphs become uninteresting for larger sizes, as most of their feasible solutions tend to assume a specific structure that is easy to optimise. We provide a conjecture on a possible reason for this phenomenon, based on the theoretical properties of connectivity in random graphs. Finally, in order to stimulate the development and testing of future approaches, we also introduce harder instances, given by small-world, regular, planar, grid and real-world graphs from the clustering literature. For all benchmark instances, we provide in the supplementary material the best results obtained by the proposed algorithms. The empirical analysis of these results suggests that in most cases good solutions for the *WSSP* tend to be connected.

Section 2 provides the survey on the relevant literature. Section 3 describes the two *GRASP* metaheuristics, while the approaches with delayed termination are discussed in Section 4. Section 5 reports the computational results.

2. Survey of the literature

From the theoretical point of view, the problem has been deeply investigated since its first appearance in Fujita et al. (2016), which proved its strong $\mathcal{NP}$-hardness, even in the unweighted case ($w_v = 1$ for all $v \in V$). In the following years, a stream of theoretical papers have characterised the complexity of the problem for special graph topologies and weight functions. The *WSSP* is polynomial on paths (Bapat et al., 2016). It is pseudopolynomial on bounded-treewidth graphs and interval graphs, polynomial on unweighted trees, $\mathcal{NP}$-hard on unweighted planar graphs and split graphs (Àgueda et al., 2018). Finally, it is fixed-parameter tractable on general unweighted graphs, with a parameter depending on the size of the solution or the neighbourhood diversity (Belmonte et al., 2020).

From the computational point of view, a number of exact algorithms have been proposed to solve instances with few dozens of vertices. Two branch-and-cut approaches, based on Integer Linear Programming (*ILP*) formulations, respectively require a cubic (Macambira et al., 2019) or a linear (Malaguti and Pedrotti, 2023) number of binary variables and an exponential number of constraints, with respect to the number of vertices of the graph. A compact Mixed Integer Linear Programming (*MILP*) formulation (Hosteins, 2020) uses a quadratic number of binary and real variables and a cubic number of constraints. Finally, the combinatorial branch-and-bound algorithm of Boggio Tomasaz et al. (2022) solves most of the available benchmark instances. To the best of our knowledge, the only heuristic approach in the literature is the randomised destructive metaheuristic that is used to provide upper bounds for the first branch-and-cut approach (Macambira et al., 2019) and later exploited also by Malaguti and Pedrotti (2023). This heuristic starts from the whole vertex set, that is trivially feasible, and iteratively removes one vertex at a time. It chooses the removed vertex at random, favouring those which minimise a combination of the degree of the vertex and the sum of the weights of the adjacent vertices previously removed from the solution. The intuition behind this choice is that such vertices reduce the risk to leave large and heavy connected components outside the solution.

3. Two *GRASP* metaheuristics

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

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

Algorithm 1 provides the pseudocode of the *GRASP* metaheuristics we propose for the *WSSP*. They start from the empty set and progressively extend it: each iteration of the loop in lines $3-6$ adds an unsafe vertex v_{new}, until all safety constraints are satisfied. Procedure `Extraction()` randomly chooses the new vertex with a selection criterium discussed in the following, that favours vertices of large degree. Since the final solution can include redundant vertices, the auxiliary function `Destructive()` turns it into a minimal one, testing the removal of each vertex in nonincreasing weight order, and performing it when the result is feasible (lines $10-16$). Vertices of equal weight are considered by nondecreasing degree, following the same intuition of the construction phase.

3.1. Selection criterium

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

input : $G = (V, E)$: connected undirected graph

$w : V \to \mathbb{Q}^+$: weight function on the vertices

$\mu \in [0, 1], \alpha \in [0, +\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters

output: S: solution returned by the algorithm

Algorithm `GRASP`(G, w, μ, α, *type*)**:**

  $S := \emptyset$

  while S *is not a safe set* **do**

    $v_{new} :=$ `Extraction`(G, S, $V \setminus S$, μ, α, *type*)

    $S := S \cup \{v_{new}\}$

  end

  return `Destructive`(G, w, S)

Function `Destructive`(G, w, S)**:**

  $Q := S$

  while $Q \neq \emptyset$ **do**

    $v_{max} := \arg\max_{v \in Q} w(v)$ `// minimise the degree to break ties`

    $Q := Q \setminus \{v_{max}\}$

    if $S \setminus \{v_{max}\}$ *is a safe set* **then**

      $S := S \setminus \{v_{max}\}$

    end

  end

  return S

Algorithm 1: The Greedy Randomised Adaptive Search Procedures

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

Definition 1. *Given a connected undirected graph $G = (V, E)$ and a subset $S \subseteq V$, let the **unsafe degree** of a vertex $v \in V$ be the number of adjacent vertices belonging to $V \setminus S$.*

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

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

3.2. Randomisation schemes

Algorithm 2 reports the pseudocode of the `Extraction()` procedure, that selects the new vertex to add to S applying one of two alternative randomisation mechanisms. Notice that the candidate set C from which the vertex is extracted coincides with $V \setminus S$ in the *GRASP* metaheuristics, but it will be reduced in the following algorithms.

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

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

In summary, the two *GRASP* metaheuristics combine a randomised constructive phase, based on the vertex unsafe degree and focused on feasibility, and a deterministic destructive phase, based on the weight of the vertices and focused on optimality. The latter can be interpreted also as a local search improvement method.

input : $G = (V, E)$: connected undirected graph

S: current set of safe vertices

C: candidate set of unsafe vertices

$\mu \in [0, 1], \alpha \in [0, +\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters

output: v_{new}: vertex selected to be added to S

Function `Extraction`(G, S, C, μ, α, *type*)**:**

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

  if *type* = *RCL* **then**

    $\delta_{\min} := \min_{v \in C} \delta^{V \setminus S}(v)$

    $\delta_{\max} := \max_{v \in C} \delta^{V \setminus S}(v)$

    $\delta_{\mu} := \mu \cdot \delta_{min} + (1 - \mu) \cdot \delta_{max}$

    $C' := \left\{v \in C \mid \delta^{V \setminus S}(v) \geq \delta_{\mu}\right\}$

    forall $v \in C'$ **do** $\pi(v) := 1/|C'|$

    forall $v \in C \setminus C'$ **do** $\pi(v) := 0$

  else if *type* = *HBSS* **then**

    forall $v \in C$ **do** $\varphi(v) := (\delta^{V \setminus S}(v))^{\alpha} + 1$

    forall $v \in C$ **do** $\pi(v) := \varphi(v) / \sum_{x \in C} \varphi(x)$

  end

  $v_{new} \leftarrow$ random extraction from C with probability $\pi(\cdot)$

  return v_{new}

Algorithm 2: The randomised selection procedure

In fact, consider the neighbourhood that includes the feasible solutions obtained adding or removing a single vertex. A steepest descent algorithm based on it would behave as the destructive procedure: it would always select the vertex of maximum weight that can be feasibly removed from the solution. Larger neighbourhoods could be more effective, but the combination of weight and connectivity aspects of the safety constraints would probably make their exploration quite inefficient.

The complexity of the *GRASP* metaheuristics can be easily computed based on the pseudocode of Algorithms 1 and 2. The constructive phase calls for at most $|V|$ times the `Extraction()` procedure, whose complexity is $T_{\text{Extr}} \in O(|E| + |V|)$, because it scans the $|E|$ edges of the graph to compute $\delta^{V \setminus S}$ and the $|V|$ vertices to determine their probabilities and select one. The destructive phase has complexity $T_{\text{Destr}} \in O(|V|(|E| + |V|))$, because it determines which of the $O(|V|)$ vertices of S can be feasibly removed by visiting the induced subgraph in $O(|E| + |V|)$ time. Overall, the computational time is $T_{\text{GRASP}} \in O(|V|(|E| + |V|))$.

4. A heuristic with delayed termination

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

4.1. Simple delayed termination

The pseudocode of the *SDT* heuristic is reported in Algorithm 3. It starts from the feasible solution generated by the *GRASP* metaheuristic (lines $2-7$). Then, it further enlarges the solution, performing a number $\gamma|V|$ (with $\gamma \in [0, 1]$) of extra iterations proportional to the number of vertices. When $\gamma = 0$, the *SDT* heuristic reduces to the *GRASP* algorithm.

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

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

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

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

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

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

4.2. Avoidant delayed termination

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

Algorithm 4 provides a detailed pseudocode. The *ADT* heuristic starts from the empty set and initialises the best known solution as the whole vertex set. At each iteration, it performs an exploration phase that considers all unsafe vertices as candidates and tests for each of them whether adding it to S yields a feasible solution. If it does, the solution is improved by the destructive procedure and compared with the best known one, in order to keep it updated, and the vertex is removed from the candidates. After the exploration, one of the remaining candidate vertices is extracted at random, with the randomisation schemes already described in Section 3.2, and added to the current solution. If all extensions of the current set are feasible, the candidate set is empty

input : $G = (V, E)$: connected undirected graph

$w : V \to \mathbb{Q}^+$: weight function on the vertices

$\mu \in [0, 1], \alpha \in [0, +\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters

$\gamma \in [0, 1]$: delay factor

output: S^*: solution returned by the algorithm

Algorithm `SDT(`G, w, μ, α, *type*, γ`)`**:**

$S := \emptyset$

while S *is not a safe set* **do**

$v_{new} :=$ `Extraction(`G, S, $V \setminus S$, μ, α, *type*`)`

$S := S \cup \{v_{new}\}$

end

$S^* :=$ `Destructive(`G, w, S`)`

for $l = 1, \ldots, \gamma |V|$ **do**

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

$v_{new} := \arg\min_{v \in C} w_v$ `// maximise the degree to break ties`

$S := S \cup \{v_{new}\}$

$S' :=$ `Destructive(`G, w, S`)`

if $w(S') < w(S^*)$ **then** $S^* := S'$

end

return S^*

Algorithm 3: The Simple Delayed Termination heuristic

and the algorithm terminates. Otherwise, the termination is delayed, as in the *SDT* heuristic, until the number of extra vertices added after finding the first feasible solution exceeds $\gamma|V|$. Notice that, even when $\gamma = 0$, the algorithm does not reduce to the *GRASP* metaheuristic, because the additional exploration phase in general finds several feasible solutions, instead of a single one.

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

4.3. Truncated avoidant delayed termination

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

Algorithm 5 provides the pseudocode. The *TADT* heuristic starts from an empty set, initialises the best known solution with the whole vertex set and considers all unsafe vertices as candidates. Instead of scanning them systematically, it extracts one candidate at random with the already described randomisation schemes, and checks whether adding it to the current solution makes it feasible. If it does, the algorithm improves the solution, possibly updates the best known one and proceeds to another extraction, removing the used vertex from the candidate set (and, of course, not adding it to the solution), as in the regular *ADT*. If the solution obtained is not feasible, however, the algorithm straightly proceeds with the next iteration, ignoring all candidates left. This reduces the computational effort, while possibly missing feasible solutions. The rationale is that, since the extraction is biased in favour of the more promising vertices, the unexplored solutions are likely to be also infeasible, or at least less interesting than the explored ones. This approach is similar in principle to the so called *first-best* exploration strategy of large neighbourhoods, that accepts the first improving neighbour solution found, as opposed to the *global-best* strategy, that selects the best one (Hansen and Mladenović, 2006).

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

input : $G = (V, E)$: connected undirected graph

$w : V \to \mathbb{Q}^+$: weight function on the vertices

$\mu \in [0, 1], \alpha \in [0, +\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters

$\gamma \in [0, 1]$: delay factor

output: S^*: solution returned by the algorithm

Algorithm `ADT(`G, w, μ, α, *type*, γ`)`**:**

$S := \emptyset$

$S^* := V$

$l := 0$

found := `false`

repeat

  $C := V \setminus S$

  for $c \in V \setminus S$ **do**

    if $S \cup \{c\}$ *is a safe set* **then**

      found := `true`

      $S' :=$ `Destructive(`$G, w, S \cup \{c\}$`)`

      if $w(S') < w(S^*)$ **then** $S^* := S'$

      $C := C \setminus \{c\}$

    end

  end

  if $C = \emptyset$ **then break**

  $v_{new} :=$ `Extraction(`G, S, C, μ, α, *type*`)`

  $S := S \cup \{v_{new}\}$

  if found **then**

    $l := l + 1$

  end

until $l > \gamma |V|$

return S^*

Algorithm 4: The Avoidant Delayed Termination heuristic

explores other feasible solutions, terminating only at the first infeasible one.

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

5. Computational experiments

This section describes the computational results obtained by the algorithms presented above on a large collection of benchmark instances with different sizes, topologies and weight functions. All algorithms have been coded in C99, and compiled with `GNU GCC 8.3.0`. They have been run on a Linux server, with processor Intel Xeon E5-2620 2.1GHz server with 16GB of RAM.

After discussing the features of the benchmark instances employed (Section 5.1), we will present the tuning of the parameters for the *GRASP* metaheuristics (Section 5.2) and for the delayed termination algorithms (Section 5.3). The performance of all algorithms on random graphs will show unexpected results, suggesting that these instances are particularly easy to solve. We will also propose a conjecture for the motivations of this behaviour. Then, we will compare the performance of these algorithms with each other (Section 5.4) and the performance of the best one with the only existing heuristic in the literature and with the known optimal values (Section 5.5). Finally, we will discuss the structure of the solutions, focusing in particular on the number of safe and unsafe components (Section 5.6).

Unfortunately, optimal results are known only for instances up to 60 vertices, while for larger instances the quality of the lower bound is bad. Consequently, the results will be evaluated computing the percent gap $(z-z^*)/z^*$, where z is the value obtained by each specific algorithm and z^* the best known value for the same instance. This underestimates the actual optimality gap and the dissimilarities between different parameter tunings and algorithms, but is the best possible measure with the available information.

5.1. Instances

The benchmarks considered in our computational experiments are available at `https://homes.di.unimi.it/cordone/research/wssp.html`, together with the detailed results. They can be grouped into several classes based on their topology, size and weight function. The first two classes derive from the literature: they consist of small instances, and will be only used in Section 5.5 to compare the proposed algorithms with the only existing heuristic approach in the literature, and to assess its capability to attain the optimal solution in short time.

Benchmark `M.` (Macambira et al., 2019) This benchmark consists of Erdős-Renyi graphs with a number of vertices covering all values from 10 to 30. For each size, three different graphs are obtained setting the number of edges to $\lfloor \delta n(n-1)/2 \rfloor$, with density $\delta \in \{0.3, 0.5, 0.7\}$. Each graph corresponds to a weighted instance,

input : $G = (V, E)$: connected undirected graph

$w : V \to \mathbb{Q}^+$: weight function on the vertices

$\mu \in [0, 1], \alpha \in [0, +\infty)$, type $\in$ {RCL,HBSS}: randomisation parameters

$\gamma \in [0, 1]$: delay factor

output: S^*: solution returned by the algorithm

Algorithm `TADT`(G, w, μ, α, *type*, γ)**:**

$S := \emptyset$

$S^* := V$

$l := 0$

found := `false`

repeat

  $C := V \setminus S$

  stop := `false`

  while not *stop* **and** $C \neq \emptyset$ **do**

    v_{new} := `Extraction`(G, S, C, μ, α, *type*)

    $C := C \setminus \{v_{new}\}$

    if $S \cup \{v_{new}\}$ *is a safe set* **then**

      found := `true`

      S' := `Destructive`$(G, w, S \cup \{v_{new}\})$

      if $w(S') < w(S^*)$ **then** $S^* := S'$

    else

      $S := S \cup \{v_{new}\}$

      if found **then**

        $l := l + 1$

      end

      stop := `true`

    end

  end

until $l > \gamma |V|$ **or** $C = \emptyset$

return S^*

Algorithm 5: The Truncated Avoidant Delayed Termination heuristic

with weights randomly extracted from a uniform distribution in $\{1, \ldots, 100\}$, and an unweighted one, where all weights are unitary. Overall, therefore, this benchmark consists of $21 \cdot 3 \cdot 2 = 126$ instances. It is publicly available at `http://www.cos.ufrj.br/~luidi/papers/safeset.html`. The optimal values of these instances are known.

Benchmark `H`. (Hosteins, 2020; Boggio Tomasaz et al., 2022) This benchmark has a similar structure, as it also includes Erdős-Renyi graphs, but the number of vertices is $n \in \{20, 25, 30, 35, 40, 50, 60\}$, there are four classes of densities ($\delta \in \{0.1, 0.2, 0.3, 0.4\}$) and 5 instances for each size and density. Also in this case, weighted and unweighted versions of the same graphs are considered, but in the former the weights are randomly extracted from a uniform distribution in $\{1, \ldots 10\}$. Overall, this benchmark consists of $7 \cdot 4 \cdot 5 \cdot 2 = 280$ instances. The optimal values are known for most of these instances.

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

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

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

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

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

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

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

Table 1: Main features of the benchmark instances: acronym, type, number of instances (#), range of the number of vertices (n), average degree ($\bar{\delta}$) and average diameter (ϕ). The upper two benchmarks derive from the literature, the other ones are new.

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

Overall the number of instances considered is $126 + 280 + 200 + 18 + 100 + 100 + 50 + 20 = 894$. Table 1 summarises their basic features, reporting for each benchmark the name, the type and number of the instances, the range of the number of vertices n, the average degree $\bar{\delta}$ and the average diameter ϕ.

5.2. Parameter tuning for the GRASP *metaheuristics*

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

We have first applied the 11 resulting heuristics (one for each parameter value considered: 7 for *RCL* and 4 for *HBSS*) with a time limit of 10 seconds on the large random graphs of benchmark H^+. The outcome was completely unexpected: on a large majority of instances (140 out of 200) all heuristics returned the same result. What is more, this result is the best one found by all algorithms, including also the large number of *SDT*, *ADT* and *TADT* variants discussed in the following. The phenomenon shows a clear trend with respect to the size, density and weight distribution of the instances. Table 2 reports for each weight function (on the

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

Table 2: Maximum gap of the solutions returned by the 11 different parameter settings of the *GRASP* metaheuristics on each group of 5 instances of benchmark $\texttt{H}^+$ with the same number of vertices, density and weight distribution

left the weighted instances, on the right the unweighted ones), size (on the rows) and density (on the columns) the maximum gap obtained by any of the 11 *GRASP* heuristics on any of the 5 instances with the given features. Label "-" marks the classes in which all settings obtained the best result for all instances: they are all characterised by a large size $|V|$ and density δ. The convergence of these values to zero is sharp, and quicker for the unweighted instances than for the weighted ones.

Our conjecture is that these results are optimal, but we have no formal proof for this statement. Hypothetically, an unknown flaw could affect all algorithms and induce them to generate the same result, but only if the instance is sufficiently large and dense. What makes this explanation unlikely is that the solution systematically returned has nearly always the same structure: the safe set forms a single component with a value equal to half the total weight of the graph. Notice that the constructive phase of all the approaches considered adopts as a selection criterion the unsafe degree of the vertices, that neglects the edges between each vertex and the current partial solution. Such a criterion favours the choice of vertices that are nonadjacent. In fact, the heuristics typically start with a disconnected subset of vertices, that only gradually merge as new ones are added. So, the algorithms are actually biased against building a single safe component. Moreover, there is a second reason in favour of assuming that such solutions are indeed optimal. In fact, it is a known property of random graphs of fixed density that their connectivity progressively increases with size, making it harder to keep a large number of vertices disconnected from each other (Frieze and Karoński, 2015). Consequently, any safe set with a weight smaller than half the weight of the graph tends to be infeasible, because it faces a complementary set of unsafe vertices that has a larger total weight and tends to be connected.

Table 3 reports the average gaps achieved by the *GRASP* metaheuristics with their 11 different tunings on the large benchmarks. The upper half refers to the weighted instances and the lower half to the unweighted ones. Each row concerns a class of instances (plus a row with the overall average), each column a different tuning of the randomisation parameter, with the *RCL* mechanism on the left and the *HBSS* one on the right. We provide aggregate results because no interesting dependency appears with respect to the size or density of

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

Table 3: Average gaps produced by the different versions of the GRASP metaheuristic for each class of instances

the instances.

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

5.3. Parameter tuning of the delayed termination approaches

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

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

We therefore consider the same values of μ and α used above, while γ is set in $\{0.0, 0.1, 0.2, 0.3, 0.4\}$. Tables 4, 5 and 6 report the overall average gaps with respect to the best known result on the large benchmarks `H+`, `SW`, `Reg`, `Pla` and `Grid`, respectively for the *SDT*, *ADT* and *TADT* heuristics. The upper half of the tables is dedicated to the weighted instances, the lower half to the unweighted ones. The left part considers the 7 possible values of parameter μ for the *RCL* mechanism, the right part the 4 values of α for the *HBSS* mechanism. Each row refers to one of the 5 values of the delayed termination parameter γ.

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

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

Concerning *ADT* (Table 5), the best average gap, that is 9.29%, is obtained by $\mu = 0.2$ and $\gamma = 0.4$, and the maximum one is equal to 49.11%. The unweighted instances behave in a similar way, with the best tuning in $\mu = 0.3$ and $\gamma = 0.4$, yielding an average gap equal to 7.82% and a maximum one equal to 52.94%. These gaps are much larger than those obtained by *SDT*. This could suggest that keeping on the border of the

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

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

feasible region is a less effective strategy than diving into it, but it could also simply be due to the stronger computational cost of *ADT*. The *RCL* mechanism tends to be better than *HBSS*: nearly all good parameter settings adopt the former.

TADT (Table 6) applies the same rationale as *ADT* while limiting the computational time and, indeed, it obtains better gaps. For the weighed instances, the best average gap, that is 6.90%, is obtained by $\mu = 0.3$ and $\gamma = 0.2$, while the maximum one is 41.38%. For the unweighted instances, the best tuning is $\mu = 0.3$ and $\gamma = 0.1$ or 0.2, with an average gap equal to 5.06% and a maximum one of 52.94%. These settings are not only similar to each other, but also to those used by *SDT*. Moreover, as for *SDT*, the *RCL* mechanism dominates the *HBSS* one.

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

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

Table 5: Average gaps produced by the different versions of the ADT heuristic on the large instances of benchmarks `H+`, `SW`, `Reg`, `Pla` and `Grid`. In dark grey the settings for which the p-value of these tests is ≥ 0.05 and in light grey those for which $0.05 > p \geq 10^{-4}$.

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

Table 6: Average gaps produced by the different versions of the $TADT$ heuristic on the large instances of benchmarks `H+`, `SW`, `Reg`, `Pla` and `Grid`. In dark grey the settings for which the p-value of these tests is ≥ 0.05 and in light grey those for which $0.05 > p \geq 10^{-4}$.

5.4. Comparison of the competing approaches

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

In particular, for the weighted instances we compare *GRASP* with $\mu = 0.2$, *SDT* with $\mu = 0.3$ and $\gamma = 0.2$, *ADT* with $\mu = 0.2$ and $\gamma = 0.4$ and *TADT* with $\mu = 0.3$ and $\gamma = 0.2$, once again with a running time of 10 seconds on each instance. The average gaps reported in Table 4 suggest that *SDT* should outperform *TADT*, whereas the other approaches should be worse than the previous ones, but more or less comparable with each other. A more detailed comparison can be performed with the *SQD* diagram (Hoos and Stützle, 2004), that reports the fraction of instances for which the gap achieved does not exceed each possible value. Figure 1 provides the diagram of the four algorithms on the weighted large instances. It shows a clear dominance of *SDT* over *TADT*, and of the latter over the other two algorithms. *GRASP* and *ADT* seem comparable, even if *GRASP* has worse results. Wilcoxon's test fully confirms these feelings: while the difference between *GRASP* and *ADT* is not statistically significant ($p \leq 0.083$ in favour of *GRASP*), all other relations have a p-value smaller than 10^{-11}. The number of best known results found is consistent with these results: 76 for *SDT*, 62 for *TADT*, 59 for *GRASP* and *ADT*.

As for the unweighted instances, we compare *GRASP* with $\mu = 0.4$, *SDT* with $\alpha = 1$ and $\gamma = 0.1$, *ADT* with $\mu = 0.3$ and $\gamma = 0.4$ and *TADT* with $\mu = 0.3$ and $\gamma = 0.2$. Figure 2 shows the corresponding *SQD* diagram. In this case, *GRASP* dominates *ADT* and both are outperformed by *SDT* and *TADT*, which, however, do not clearly dominate each other. *SDT* seems more conservative, having less solutions with a large gap, but *TADT* finds a larger fraction of solutions with a small gap (less than 5%). According to Wilcoxon's test, *ADT* is significantly dominated by the other three algorithms ($p < 10^{-4}$) and *GRASP* is dominated by *TADT* ($p \leq 3.55 \cdot 10^{-5}$). The comparison of *SDT* with *GRASP* is weakly in favour of the former ($p = 0.06$), whereas we cannot say anything about *SDT* and *TADT*. As for the number of best know results found, this is slightly in favour of *TADT* (106) over *SDT* (96), while *GRASP* and *ADT* are worse, and nearly equivalent to each other (respectively, 87 and 86).

The wide range of gaps observed reflects the very different performance that the algorithms exhibit on different classes of instances. This phenomenon is more pronounced in the *GRASP* metaheuristic than in the delayed termination algorithms, but clear in all cases. Table 7 describes in more detail the average gaps with respect to the best known results obtained by each algorithm on specific classes of instances. The left part of the table refers to the weighted instances, and the right part to the unweighted ones. In each part, four columns correspond to the four competing algorithms. A group of rows is associated with each benchmark, and further disaggregated with respect to a relevant structural parameter: benchmark `H+` is divided according to the density of the graph, benchmarks `SW` and `Reg` according to the (respectively, initial or exact) vertex degree, benchmark

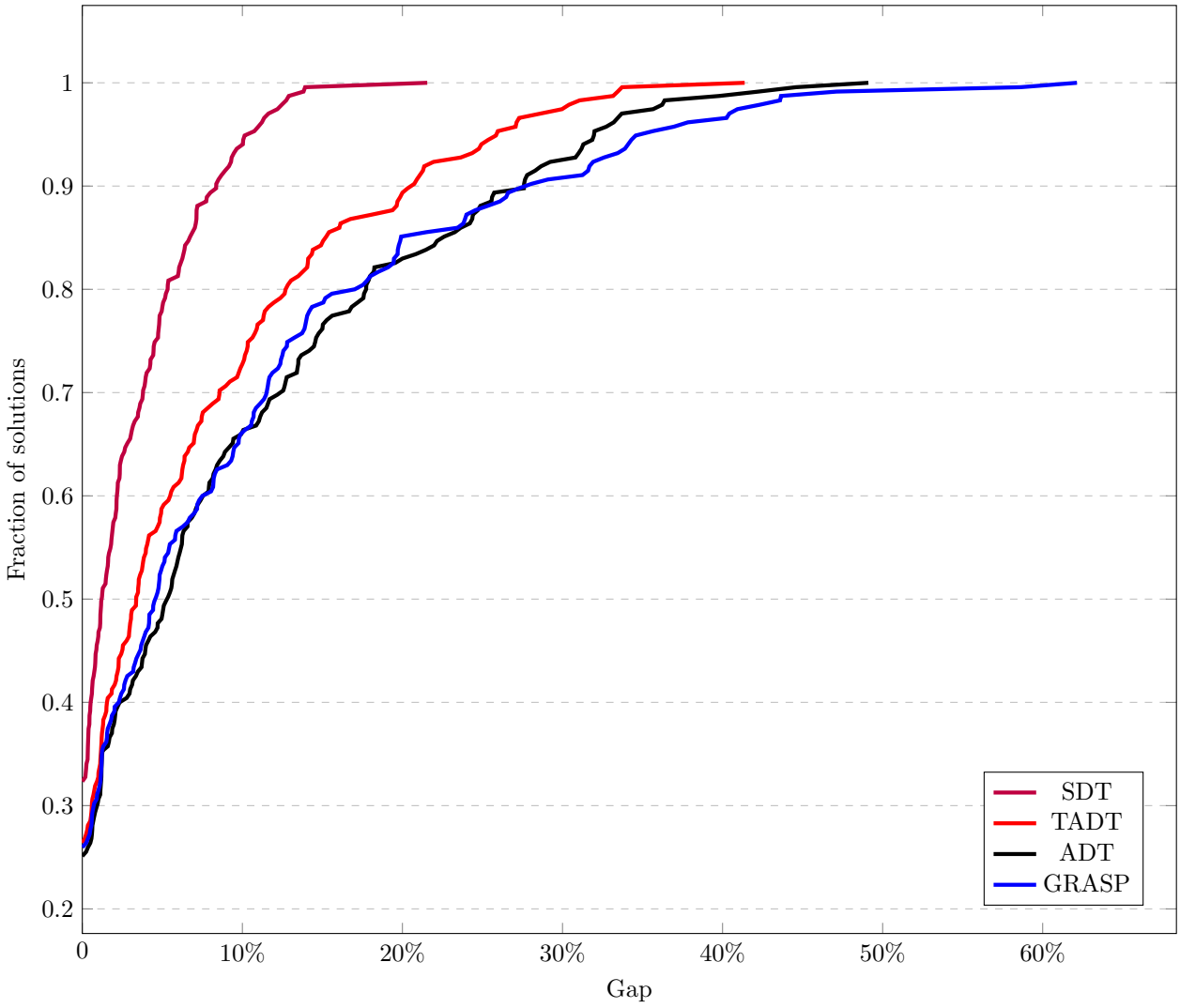

Figure 1: Solution Quality Distribution diagram over the large weighted instances of the presented heuristics (each using its best parameters). The diagram reports on the y-axis the fraction of instances such that each algorithm returns a solution whose gap (with respect to the best solution) is not worse than the value reported on the x-axis.

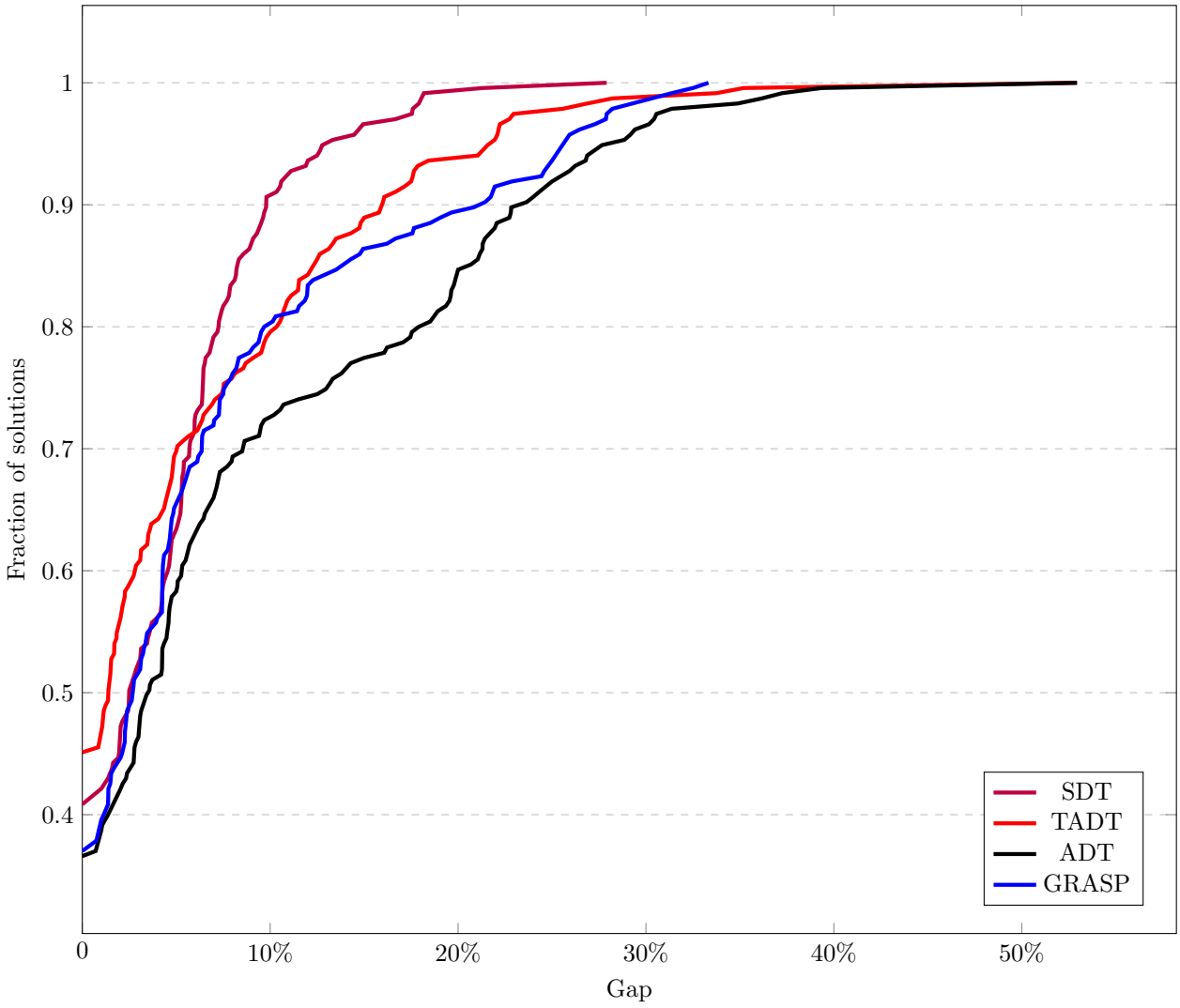

Figure 2: Solution Quality Distribution diagram over the large unweighted instances of the presented heuristics (each using its best parameters). The diagram reports on the y-axis the fraction of instances such that each algorithm returns a solution whose gap (with respect to the best solution) is not worse than the value reported on the x-axis.

Grid is divided into 2D and 3D grids. We can see that *SDT* finds the best average gaps for most instance classes, but is less effective on the weighted grids and the unweighted regular graphs. Conversely, *TADT* has a good performance on these two classes. *ADT* and *GRASP* are less effective, apart from some classes of unweighted random graphs. This is consistent with the plots of Figures 1 and 2.

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

Table 7: Average gaps produced by each heuristic (using its best parameters) on specific subsets of the large benchmark instances.

Since the real-world graphs of benchmark GC are rather heterogeneous, we analyse them individually. The first three columns of Table 8 provide the name of each instance, its number of vertices $|V|$ and of edges $|E|$. The following two blocks of four columns report, respectively, for the weighted and the unweighted instance, the value of the objective function for the solution returned by each of the four competing algorithms. The best result for each instance is bolded. The last row of the table provides the average percent gap with respect to the best known result. While the smaller instances are solved with the same value by all algorithms, the larger ones show significant differences. The *TADT* heuristic hits the largest number of best known results (11 out of 18) and has the smallest average gap (0.92%), but in this benchmark *GRASP* comes second (with 10 best results and a gap equal to 2.17%) and the other two heuristics are worse, in particular in the unweighted case.

This is in contrast with the better performance of *SDT*, observed on average for the other benchmarks, but consistent with the good performance of *TADT*.

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

Table 8: Results obtained by the presented heuristics (with their best parameter tuning) in 10 seconds on the real-world graphs of benchmark `GC`

5.5. Comparison with the state of the art

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

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

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

Table 9 reports the results aggregated for instances with the same weight function and graph density δ, as specified in the first two columns. The third column provides the number of instances for each class.

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

Table 9: Results obtained by 1000 iterations of *RD* and *SDT* on benchmark `M`. We display the number of instances for which one of the two algorithms outperforms the other, the number of optimal values found, the average and maximum gap with respect to the optimum and the (average and minimum) ratio of the computational times.

The following block of two columns (labelled "Domination") displays the number of instances for which each algorithm yields a result strictly better than its competitor. Then, the table provides the number of optima found by the two algorithms (all instances are solved exactly). The average and maximum gaps follow (with the better one in bold). The last block of two columns reports the average and the minimum value of the ratio between the computational times of *RD* and *SDT*.

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

Since benchmark `H` has been solved only with exact algorithms, and the performance of a heuristic cannot be meaningfully compared with them, we evaluate the ability of *SDT* to find in short time the optimal or best known results. In particular, the branch-and-bound algorithm of Boggio Tomasaz et al. (2022) solves to optimality 254 out of the 280 instances, but requires one hour of computation on the same machine. Table 10 reports the results of *SDT* with a time limit of 60 seconds: each row corresponds to one of the possible sizes and each column to one of the four classes of density. The left part of the table concerns the weighted instances, and the right part the unweighted ones. Each cell contains the average gap with respect to the optimal, or the best

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

Table 10: Results obtained by *SDT* in 60 seconds on the small random instances of benchmark H.

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

5.6. Structure of the solutions and relation with the Connected Safe Set Problem

The literature on the *SSP* often discusses a variant, known as *Connected Safe Set Problem* (*CSSP*), in which the solution is required to consist of a single safe component (Fujita et al., 2014). We have already remarked in Section 5.2 that most of the best known solutions for random graphs consist of a single safe component opposed to a single unsafe one. We here extend the analysis of the structure of the best solutions to all large benchmarks.

Table 11 considers the same classes of instances addressed by Table 7. The first two columns specify the benchmark and class. The following two blocks of four columns refer, respectively, to the weighted and the unweighted instances. In each block, two columns, labelled $|\mathcal{C}_G(S)|$, provide the average and the overall range of values (minimum and maximum between square brackets) that the number of safe components assumes in the best known solution. The other two columns, labelled $|\mathcal{C}_G(V \setminus S)|$, provide the same information for the unsafe components. We refer to the best known solutions because they are the only possible approximation of the unknown optimal ones.

The clearest observation is that a large majority of solutions have a single safe component (445 out of 470), while 24 have 2 components and only one has 3. In short, the *CSSP* has very frequently the same solution as the

basic *SSP*. Considering the unsafe components, the random instances exhibit the behaviour already described in Section 5.2: the best known solution is often complemented by a single unsafe component, and this becomes more and more frequent as the size and density of the instances increases. The sparse instances show a wider range of values, but still the number of unsafe components is often equal to 1 (in 140 instances out of 470) and most of the time less than 10 (in more than 400 instances). The larger values seem to occur in the regular instances, but it is presently impossible to determine whether this actually reflects the structure of the optimal solutions for such instances, or the fact that the best known results are actually far from optimal. The same results are illustrated in Figure 3 which displays the number of safe components (in blue) and unsafe components (in red) for each of the 235 large instances. In order to improve readability, the instances have been sorted in ascending order of the number of safe components (in the first case) or of the number of unsafe components (in the second case) in the best known solution. The figure can also be understood as the cumulative number of instances for which the best known solution has a number of safe (or unsafe) components lower than a threshold given on the vertical axis.

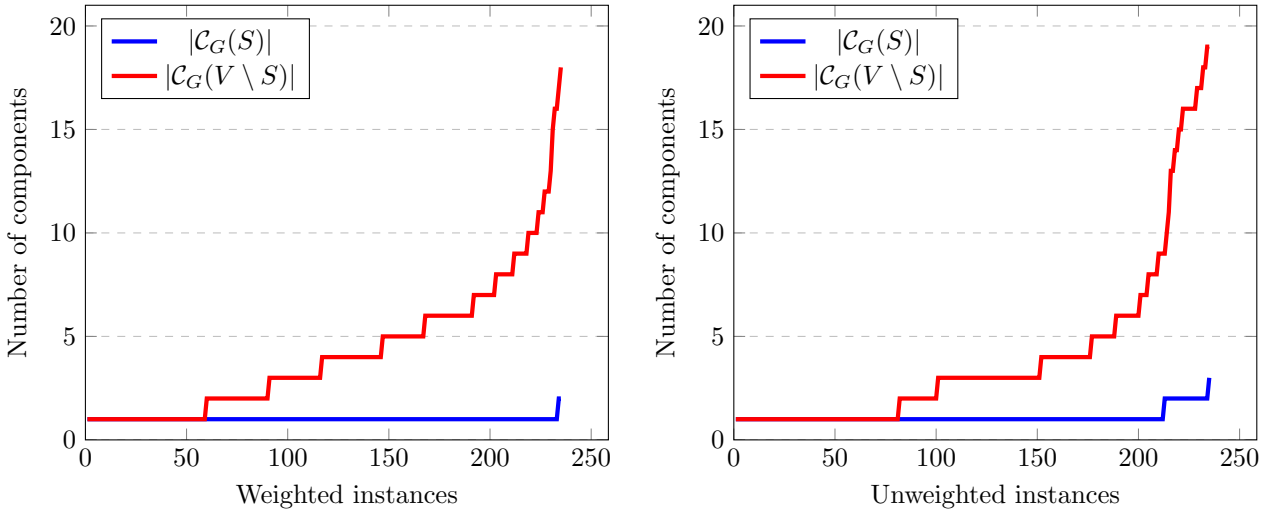

Figure 3: Number of safe and unsafe components in each instance, for both weighted (left) and unweighted instances (right).

6. Conclusions

This paper tackles the *WSSP*, an $\mathcal{NP}$-hard graph optimisation problem with applications to social network analysis and emergency facility design. We have developed *GRASP* metaheuristics, based on a randomised constructive procedure and a deterministic destructive procedure that, respectively, take into account the feasibility and the optimality of the solutions obtained. These metaheuristics apply the two common randomisation

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

Table 11: Average and range ([minimum,maximum]) of the number of safe and unsafe components ($|\mathcal{C}_G(S)|$ and $|\mathcal{C}_G(V \setminus S)|$) in the best known solutions for different classes of instances.

schemes (that is, a uniform sampling from a restricted candidate list or a biased one from all possibilities). We have then extended these algorithms with two mechanisms to delay their termination after finding the first feasible solution, one that dives into the feasible region (SDT), and one that keeps on its border (ADT). A truncated variant of the latter ($TADT$) has also been implemented and investigated. The computational experiments on a new benchmark of dense and sparse instances ranging from 100 to 300 vertices suggest that the SDT heuristic has the best performance, but the $TADT$ heuristic performs better on some classes of instances. A comparison with the destructive approach drawn from the literature shows that SDT is more effective, while taking a comparable time. When applied to the publicly available smaller instances, it finds in a few seconds most of the best known results, and improves some of those for which the optimal value is still unknown. Finally, the experimental results suggest that in a large majority of instances, good solutions for the $WSSP$ tend to be connected, even when this requirement is not enforced, in particular for large and dense graphs.

As possible future research directions on the Safe Set Problem, one may consider alternative metaheuristic approaches that could exploit the structure of the problem. Due to the non-trivial feasibility conditions of this problem, that impose connectivity and weight dominance constraints, *local search* and *path relinking* seem unpromising directions to pursue. In fact, they would generate a large proportion of unfeasible or very expensive solutions. On the contrary, a *very large scale neighbourhood search* seems more promising, since it allows to jump from the current solution to the next one not simply by basic exchanges or small steps. Indeed, the *delayed termination* heuristics presented in this article can be interpreted as an application of similar ideas. Population-based recombination heuristics, such as a *genetic algorithm* or *scatter search*, could exploit the property that the union of two feasible solutions is also feasible: it is trivially non-minimal, but it could be improved through a destructive procedure. Finally, matheuristics could also be investigated, though they usually rely on some good (meaning *tight*) *ILP*/*MILP* formulation, which is currently not available for the problem.

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., Y.Matsui, 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.

Boggio Tomasaz, A., Cordone, R., Hosteins, P., 2022. A combinatorial branch-and-bound for the Weighted Safe Set Problem. Networks [Accepted with DOI: 10.1002/NET.22140].

Dunn, O.J., 1961. Multiple comparisons among means. Journal of the American Statistical Association 56, 52–64.

Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71.

Frieze, A., Karoński, M., 2015. Introduction to Random Graphs. Cambridge University Press.

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.

Hansen, P., Mladenović, N., 2006. First vs. best improvement: an empirical study. Discrete Applied Mathematics 154, 802–817.

Hoos, H.H., Stützle, T., 2004. Stochastic local search: Foundations and applications. Elsevier.

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

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

Malaguti, E., Pedrotti, V., 2023. Models and algorithms for the weighted safe set problem. Discrete Applied Mathematics 329, 23–34.

PassMark Software, 2022. CPU benchmarks. `https://www.cpubenchmark.net`.

Shaw, P., 1998. Using constraint programming and local search methods to solve vehicle routing problems, in: Maher, M., Puget, J.F. (Eds.), Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP '98), Springer–Verlag. pp. 417–431.

Watts, D.J., Strogatz, S.H., 1998. Collective dynamics of "small-world" networks. Nature 393, 440–442.

Wilcoxon, F., 1945. Individual comparisons by ranking methods. Biometrics 1, 80–83.