Received: 18 March 2019 | Revised: 14 January 2020 | Accepted: 31 January 2020

DOI: 10.1111/exsy.12540

SPECIAL ISSUE PAPER

Expert Systems WILEY

Finding weaknesses in networks using Greedy Randomized Adaptive Search Procedure and Path Relinking

Sergio Pérez-Peló | Jesús Sánchez-Oro | Abraham Duarte

Department Computer Science, Universidad Rey Juan Carlos, Móstoles, Spain

Correspondence
Jesús Sánchez-Oro, Department Computer Science, Universidad Rey Juan Carlos, C/Tulipán, S/N, Móstoles, Spain.
Email: jesus.sanchezoro@urjc.es

Funding information
Comunidad de Madrid, Grant/Award Number: S2018/TCS-4566; Ministerio de Ciencia e Innovación, Grant/Award Number: PGC2018-095322-B-C22

Abstract

In recent years, the relevance of cybersecurity has been increasingly evident to companies and institutions, as well as to final users. Because of that, it is important to ensure the robustness of a network. With the aim of improving the security of the network, it is desirable to find out which are the most critical nodes in order to protect them from external attackers. This work tackles this problem, named the α-separator problem, from a heuristic perspective, proposing an algorithm based on the Greedy Randomized Adaptive Search Procedure (GRASP). In particular, a novel approach for the constructive procedure is proposed, where centrality metrics derived from social network analysis are used as a greedy criterion. Furthermore, the quality of the provided solutions is improved by means of a combination method based on Path Relinking (PR). This work explores different variants of PR, also adapting the most recent one, Exterior PR, for the problem under consideration. The combination of GRASP + PR allows the algorithm to obtain high-quality solutions within a reasonable computing time. The proposal is supported by a set of intensive computational experiments that show the quality of the proposal, comparing it with the most competitive algorithm found in the state of art.

KEYWORDS

critical nodes, GRASP, metaheuristic, Path Relinking, α-separator problem

1 | INTRODUCTION

In recent years, the words cyber-attack, information leakage, invasion of privacy, etc. have become increasingly common in the media. The increase in the number of attacks on networks, as well as concerns about privacy on the Internet, have created the need for more secure, reliable and robust networks. A cyber-attack on a company involving the leakage of personal information from its customers can cause significant economic and social damage (Andersson et al., 2005). Moreover, Denial of Service and Distributed Denial of Service have become two of the most widespread attacks as a successful attack can result, for example, in the deactivation of an Internet provider's service. On the other hand, if other services depend in some way on the service under attack, a cascade down may occur, affecting a considerably high number of customers. (Crucitti, Latora, & Marchiori, 2004).

It is important to identify which are the most important nodes in a network. This is a matter of concern to both actors: the attacker and the defender. On the one hand, the former is interested in disabling these important nodes to make the network more vulnerable. On the other hand, the second one will be interested in reinforcing the protection over these nodes, applying more robust security measures. Therefore, the identification of the most critical/weakest nodes in the network is a key part of the network security system.

We define a network as a graph $G = (V, E)$, where V is the set of vertices ($|V| = n$), and E is the set of edges ($|E| = m$). A vertex $v \in V$ represents a network node, while an edge $(u, v) \in E$, with $u, v \in V$, indicates a connection or link in the network between nodes u and v. We define a

network separator as the set of vertices $S \subseteq V$ whose removal divides the network into p connected components, C_1, C_2, ..., C_p, with $p \geq 2$. It can be trivially demonstrated that $V \setminus S = \cup_{i=1}^{p}$ and $C_i \cap C_j = \emptyset$ with $1 \leq i, j \leq p$ and $i \neq j$. Therefore, each subset C_i induces a subgraph $G_i = \{V_i, E_i\}$ with $V_i = C_i$ and $E_i = \{(u, v) \in E : u, v \in C_i\}$, which is disconnected from the remaining induced subgraphs.

This work focuses on the α-separator problem (α-SP), which consists of finding the minimum set of vertices S^*, whose elimination separates the network G into different connected components with fewer than $\lceil \alpha \cdot n \rceil$ vertices in each component. It is worth mentioning that the number of resultant connected components is not relevant to this problem. The actual constraint forces the number of vertices in each component to be less than or equal to $\alpha \cdot n$, with α being an input network parameter. This problem is $\mathcal{NP}$-hard for general topology networks when $\alpha \leq 2/3$ values are considered (Feige & Mahdian, 2006).

In Figure 1a, we show an example of a network with nine nodes and 10 links. Let us assume that $\alpha = 2/3$. Then, the size of each resulting connected component must not exceed $\lceil \alpha \cdot n \rceil = 2/3 \cdot 9 = 6$ nodes. Figure 1b depicts a feasible solution S_1 for the α-SP, with a separator conformed with nodes **B**, **C**, **E** and **I**. If we remove the separator and the corresponding links, the network is divided into two connected components, $C_1 = \{\mathtt{A},\mathtt{D}\}$ and $C_2 = \{\mathtt{F},\mathtt{G},\mathtt{H}\}$, with sizes 2 and 3, respectively. Then, the objective function value for this solution is equal to 4 (i.e. the size of the separator). In Figure 1c, we represent a different solution $S_2 = \{\mathtt{A},\mathtt{B}\}$. The removal of this separator (and the associated edges) produces three connected components $C_1 = \{\mathtt{E}\}$, $C_2 = \{\mathtt{D}\}$ and $C_3 = \{\mathtt{C},\mathtt{G},\mathtt{H},\mathtt{I},\mathtt{F}\}$ that satisfy the size constraint (i.e. the size of each connected component is lower than 6). Notice that S_2 is better than the S_1 as $|S_2| < |S_1|$.

Special types of networks, such as those whose topology is a tree or a cycle, can be solved in polynomial time (Mohamed-Sidi, 2014). In that work, an approximation algorithm is also presented, based on a greedy criterion, which provides an approach ratio of $\alpha \cdot n + 1$. Nevertheless, these algorithms need to have a priori knowledge about the network topology, which is not usual in real-life problems. A different approach is described in Wachs, Grothoff, and Thurimella (2012), where the authors present a heuristic algorithm to study the separators in Autonomous Systems[1] of Internet.

The α-SP is related with other well-known optimization problems. In particular, if $\alpha = 1/n$, then it is equivalent to the *minimum vertex cover problem* (Li, Hu, Zhang, & Yin, 2016. In addition, if $\alpha = 2/3$, then it is analogous to the *minimum dissociation set problem* (see Yannakakis (1981) for more node-deleting problems). The α-SP can be considered a generalization of these problems, which were also proven to be $\mathcal{NP}$-hard in Garey and Johnson (1979). A deep study on the complexity and some approximation results are presented in Ben-Ameur, Mohamed-Sidi, and Neto (2015).

This problem has been mainly ignored from a heuristic point of view. In particular, the most competitive algorithm for solving the α-SP is based on a Markov Chain Monte Carlo method hybridized with Random Walks Lee, Kwak, Lee, and Shroff (2017). Authors in that work show the superiority of their proposal, comparing it with simple heuristics such as highest-degree-first and greedy procedures.

This paper is structured as follows. Section 2 proposes an algorithmic approach to solve this problem. Section 3 presents a post-processing method based on the Path Relinking methodology. Section 4 tests the quality of the proposed algorithm, and finally, Section 5 exposes the conclusions obtained from this research.

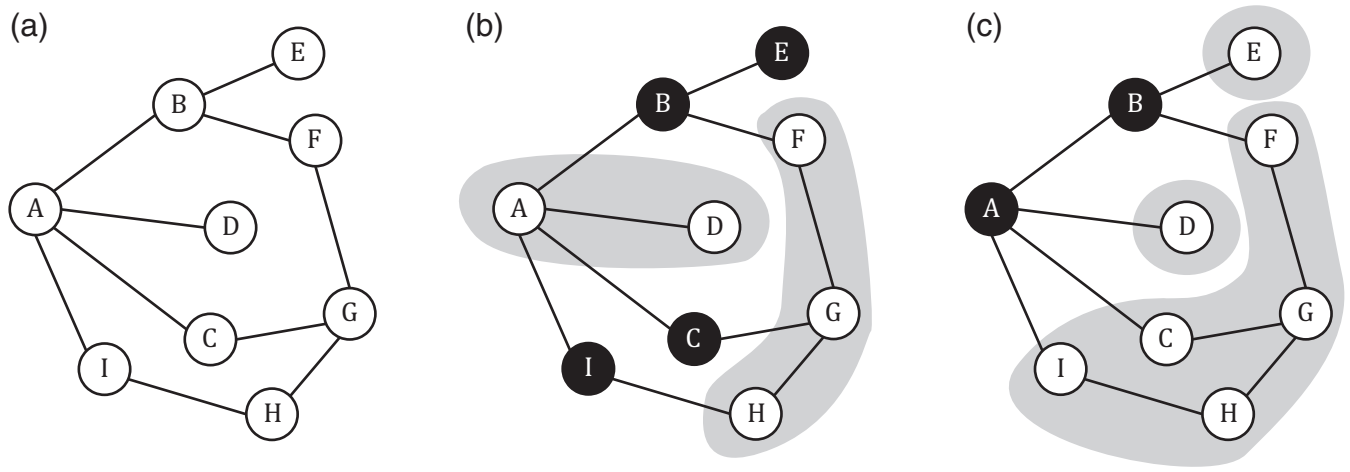

FIGURE 1 (a) Example graph derived from a network with nine nodes and ten connections between nodes, (b) a feasible solution with four nodes in the separator (**B**, **C**, **E** and **I**) and (c) a better solution with two nodes in the separator (**A** and **B**)

2 | GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE

Greedy Randomized Adaptive Search Procedure (GRASP) is a multi-start strategy that was originally proposed in Feo and Resende (1989), but it was not formally defined until Feo, Resende, and Smith (1994). As a multi-start method, GRASP tries to avoid getting trapped in local optima by restarting the search from a different starting point in the search space. This behaviour allows the algorithm to increase the diversification of the search. This metaheuristic has been recently used successfully in several optimization problems (Duarte, Sánchez-Oro, Resende, Glover, & Martí, 2015; Quintana, Sánchez-Oro, & Duarte, 2016; Sánchez-Oro, Laguna, Martí, & Duarte, 2016).

Each GRASP iteration has two distinct phases: construction and improvement. The former is designed for constructing an initial high-quality solution, while the objective of the latter is to improve the initial solution by means of a local optimization method (F. W. Glover & Kochenberger, 2006). On the one hand, constructive methods start from an empty solution and then iteratively adding elements to it. On the other hand, the local optimizer considered for the second phase is a local search method, but GRASP is commonly hybridized with more complex search methods, such as Tabu Search (F. Glover & Laguna, 1998) or Variable Neighbourhood Search (Hansen & Mladenović, 2014), among others, which highlights the versatility of GRASP methodology.

2.1 | Construction phase

The constructive procedure, named *GreedyRandom*, starts from an empty solution, and it iteratively adds nodes to the solution under construction until it becomes feasible. With the aim of diversifying the search, the first node to be inserted into the separator (i.e. the solution in the context of the α-SP) is randomly selected from *V*. Then, it is necessary to add vertices to it until the problem constraint is satisfied: the size of every connected component in the network should be smaller than $\lceil \alpha \cdot n \rceil$. Algorithm 1 presents the pseudocode of the proposed method.

Algorithm 1 *GreedyRandom*(G, β)

1: v ← *Random*(V)
2: S ← {v}
3: CL ← V \ {v}
4: **while not** *isFeasible*(S) **do**
5: $g_{min} \leftarrow \min_{v \in CL} g(v)$
6: $g_{max} \leftarrow \max_{v \in CL} g(v)$
7: $\mu \leftarrow g_{max} - \beta \cdot (g_{max} - g_{min})$
8: RCL ← {v ∈ CL : g(v) ≥ μ}
9: v ← *Random*(RCL)
10: S ← S ∪ {v}
11: CL ← CL \ {v}
12: **end while.**
13: **return** S

The procedure starts by randomly choosing the first node that will be included in the solution (steps 1–2). The random selection of this first node allows the procedure to increase the diversity of the solutions generated at the construction phase. Next, the candidate list *CL* is created with all the nodes of the network except the initially chosen one (step 3). Then, the constructive method iteratively adds a new node to the solution until it satisfies the aforementioned problem constraints (steps 4–12). The *is Feasible* function is intended to check that the size of all the connected components of the network is smaller than $\lceil \alpha \cdot n \rceil$.

The selection of the next element to be incorporated in the solution under construction is a key aspect in the design of effective GRASP constructive methods. Considering the complexity of the proposed greedy function (based on the closeness centrality metric, Newman (2008)), it will be thoroughly described in Section 2.2. For the sake of clarity, we now refer to this function as *g*. Steps 5 and 6 evaluate the minimum and maximum value of the greedy function among all the candidates, respectively. These two values are used to calculate the μ threshold (step 7) that is responsible for limiting the nodes that will be part of the restricted candidate list *RCL* (step 8). The threshold value depends on a β parameter (with $0 \leq \beta \leq 1$) that controls the randomness/greediness of the method. On the one hand, if $\beta = 0$, then $\mu = g_{max}$, and, therefore, only those vertices with the maximum value of the greedy function will be considered in the *RCL*, resulting in a totally greedy method. On the other hand, if $\beta = 1$, then $\mu = g_{min}$, allowing the selection of every node in the *CL*, becoming a totally random method. Through this way, the higher the value of the β parameter, the more random the method is and vice versa. Section 4 will discuss how this parameter affects the quality of the obtained solutions.

This work additionally proposes an alternative constructive procedure, named *RandomGreedy*, where the greedy and random phases are swapped. In the *GreedyRandom* variant, the *RCL* is created with the most promising elements of the *CL* (i.e. those whose greedy function value exceeds a certain threshold), and then, it randomly selects the next element from the *RCL*. On the contrary, the *RandomGreedy* variant first constructs the *RCL* with a set of $\beta \cdot |CL|$ elements randomly selected from the *CL*, and then, it chooses the element with the largest greedy function value among those included in the *RCL*. The main difference between the *GreedyRandom* and *RandomGreedy* constructive procedure is the interchange between the greedy and random phases of the constructive procedure. In addition, it is worth mentioning that *RandomGreedy* is presumably faster than *GreedyRandom* as it does not require the evaluation of the complete *CL* but it requires the evaluation of the elements included in the *RCL* (this hypothesis will be confirmed in Section 4).

2.2 | Greedy criterion

The selection of the next element to be incorporated in a partial solution in GRASP methodology depends on a greedy criterion. Specifically, for each node $v \in V$, it is necessary to define a greedy function $g(v)$, which evaluates the relevance of inserting it into the partial solution. In the context of the α-SP problem, the greedy criterion should indicate how likely a node is to be critical.

This work presents a novel approach to identify whether a node is critical or not. In particular, it consists of using criteria derived from social network analysis (SNA). Following this idea, the greedy criterion will consider the network a social network, identifying which nodes are relevant in the network as if it were a SNA problem. In SNA, a node is considered relevant if it is responsible for maintaining a large number of users connected through it Scott and Carrington (2011). Therefore, the analogy with the α-SP problem is straightforward: a node is important in the network if its removal disconnects several devices from the network.

The literature of SNA offers a vast amount of metrics to evaluate the relevance of a user in a social network. However, it must be kept in mind that the evaluation of most of them is rather computationally demanding. This is mainly because most of them, such as *betweenness* (Brandes, 2001) or *pagerank* (Page, Brin, Motwani, and Winograd (1999)), need to know all the shortest paths between each pair of users in the network. It is worth mentioning that the construction of all shortest paths between each pair of vertices is unapproachable for real problems. Then, it is usually tackled by considering a fixed number of the shortest paths (Yen (1971)).

In this work, we will use the metric known as *closeness* (Newman (2008)), which considers that a vertex is important if it is close to several nodes in the network. More formally, given a connected network G, the *closeness* value for a vertex is calculated as the inverse of the sum of the lengths of the shortest paths between the analysed node and the remaining nodes in the graph. It should be noted that the evaluation of this metric only requires the evaluation of the shortest path between each pair of vertices, so it is enough to make a breadth-first search (if the network is not weighted) or to apply the Dijkstra algorithm (if the network is weighted). The idea that underlies this metric is that the more central a node is, the closer to all the other nodes it is. Indeed, the larger the closeness value, the more relevant a node.

We need to adapt the general definition of closeness to the α-SP as this optimization problem deals with different connected components. It becomes a problem because the distance between nodes in different connected components is evaluated as infinite. Then, we only evaluate distances from each node v in a specific connected component C_i (with $1 \le i \le p$, with p being the number of connected components) to the remaining nodes in C_i. Specifically, the *closeness* of a node $v \in C_i$, defined as $C(v)$, is then evaluated as the inverse of the sum of the distances from that node v to the other nodes in the same connected component. In mathematical terms,

$$C(v) = \frac{1}{\sum_{u \in C_i} d(v,u)}$$

where $d(v, u)$ is the distance between nodes v and u, and C_i is the connected component to which both nodes belong. The distance between nodes v and u considered in this work is evaluated as the length of the shortest path that connects u with v in the network.

2.3 | Improvement phase

Solutions generated in the construction phase described in Section 2.1 are constructed by a randomized procedure that controls its greediness through a parameter β. For this reason, the solution constructed may not be a local optimum, and therefore, it can be further improved through any local optimization method. This work proposes a local search procedure to improve those solutions.

Local search procedures are designed with the objective of finding the local optimum of a solution with respect to a predefined neighbourhood. A neighbourhood can be defined as all the solutions that can be obtained from a single one by means of a single movement. Then, prior to defining the neighbourhood, we must define the considered movement. We define the movement $Move2x1(S, v, u, w)$ as the removal of nodes v and u from the current solution (i.e. separator) and the insertion of node w into it. More formally,

$$Move2x1(S,v,u,w) = (S \setminus \{u,v\}) \cup \{w\}$$

Notice that this movement may lead to unfeasible solutions as the resulting one might not be a separator for the problem under consideration. The proposed local search method will only consider those moves that result in a feasible solution, discarding those moves that produce unfeasible solutions. In addition, it is worth mentioning that any feasible movement will result in an improvement as the size of the separator would have been decreased in one unit.

Then, given a feasible solution S, its associated neighbourhood $N_{2\times1}(S)$ is defined as all the solutions that can be generated by applying the *Move2x1* operator to solution S, with any of the available nodes. More formally,

$$N_{2\times1}(S) \leftarrow \{S' \leftarrow Move2x1(S,v,u,w)\ \forall v,u \in S \wedge \forall w \in V \setminus S\}$$

The last step for defining a local search method consists of establishing the order in which the neighbourhood will be explored. Traditionally, two methods for exploring the neighbourhood are considered: *First Improvement* and *Best Improvement*. The former performs the first improving move found when the scanning the neighbourhood, while the latter requires the complete neighbourhood to be analysed, performing the movement that leads to the best solution. This work only considers the *First Improvement* approach, mainly due to the following reasons. First, the *First Improvement* strategy is usually less computationally demanding than *Best Improvement* because it does not explore the complete set of reachable solutions in each step but performs the first improving move found. Furthermore, note that, given the defined neighbourhood, any movement that leads to a feasible solution will produce a better solution in terms of the objective function value. Specifically, the new solution will outperform the original one in one unit as the new separator will contain one node less than the original solution. Then, the *Best Improvement* approach is discarded for this local search.

Finally, there is a need to indicate the order in which the neighbour solutions will be explored as, in the context of *First Improvement*, the order of exploration may affect the quality of the resulting solutions. The local search proposed in this work randomly explores the neighbourhood to increase the diversification of the search.

The local search method stops when no improvement is found after exploring the complete neighbourhood, returning the best solution found during the search.

3 | PATH RELINKING

Path Relinking (PR) is a solution combination method originally proposed as a methodology to integrate intensification and diversification strategies in the Tabu Search context (F. Glover, 1996). The PR behaviour is based on exploring the trajectories that connect high-quality solutions, with the objective of generating intermediate solutions that can eventually be better than the original solutions used to build the trajectory. Laguna and Martí (1999) adapted PR to the GRASP context with the aim of increasing the intensification of the search. The PR algorithm operates over a set of solutions, called *Elite Set* (ES), that is usually ordered following a predefined descendent quality criterion of each solution. In this work, we consider the value of the objective function of the solution as a quality criterion. In particular, the ES consists of the highest-quality solutions generated by the GRASP algorithm. This design is generally denominated static (F. Glover, Laguna, & Martí, 2000) because GRASP is first applied to build the ES, and then, PR is applied to explore the trajectories between all pairs of solutions in the ES. On the contrary, the dynamic design considers updating the ES each time a trajectory is explored.

A trajectory between an initial solution S_i and a guiding one S_g is generated by iteratively including attributes or properties of the guiding solution into the initial one. In the context of α-SP, we will include in the separator of the initial solution those nodes that are present in the separator of the guiding one by interchanging them with those nodes that are in the initial solution but not in the guiding one. For this purpose, we first need to define a new movement that removes from a separator S the node v and inserts node u into it, producing a new solution $S' \leftarrow (S \setminus \{v\}) \cup \{u\}$. For the sake of clarity, we named this move $S' \leftarrow Swap(S, v, u)$.

Then, a trajectory between two solutions is created by iteratively performing $Swap(S_i, v, u)$ moves, where $v \in S_i \setminus S_g$ and $u \in S_g \setminus S_i$ until S_i becomes S_g. In each iteration, PR performs a *swap* movement that generates a new intermediate solution in the trajectory. It should be noted that this movement produces, with high probability, an unfeasible solution (i.e. the intermediate solution is not a valid separator), so a repair operator must be applied to the intermediate solution to make it feasible. The proposed repair operator consists of adding random nodes to the separator until the solution becomes feasible, thus increasing the diversification of the search.

Figure 2 shows a trajectory created between two solutions, $S_i = \{B,I,G,H\}$ and $S_g = \{A,B,C,F\}$, for the graph depicted in Figure 1a, but this time considering $\alpha = 1/3$. Then, the network must be disconnected in components of $\lceil \alpha \cdot n \rceil = 1/3 \cdot 9 = 3$ nodes at most. The illustration is divided into three parts: path, which represents the actual path created between both solutions; repair, which considers those solutions of the path that have been repaired in order to make them feasible; and clean, which includes solutions of the path that originally contained redundant nodes (i.e. those that are not needed any more in the separator) where these redundant nodes have been removed. In this example, the initial solution is

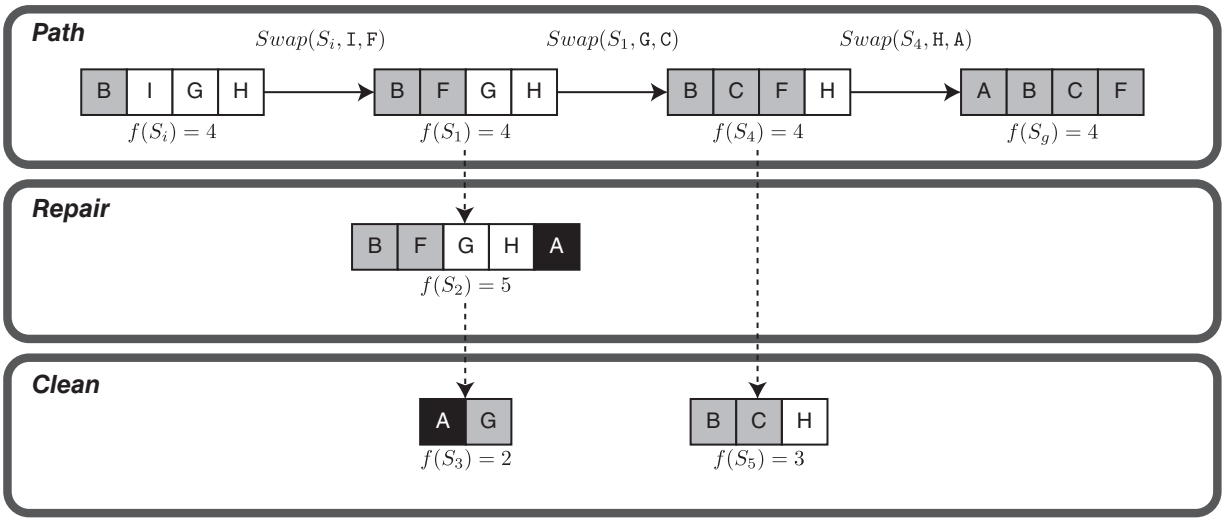

FIGURE 2 Example of a trajectory created between solutions $S_i = \{\mathtt{B},\mathtt{I},\mathtt{G},\mathtt{H}\}$ and $S_g = \{\mathtt{A},\mathtt{B},\mathtt{C},\mathtt{F}\}$, considering $\alpha = 1/3$

S_i, while the guiding one is S_g, so the trajectory consists of creating a path of solutions from S_i to S_g. In each step of the path, a new element belonging to the separator of S_g will be included in S_i, swapping it with an element included in S_i that does not belong to S_g. Figure 2 shows the nodes included in the separators of each solution, and the nodes in common with S_g have been highlighted in grey.

The first solution in the path, S_1, is generated by performing the move $Swap(S_i, \mathtt{I}, \mathtt{G})$, thus removing node **I** from the separator and replacing it with **G**. Notice that the resulting solution S_1 is not feasible as the graph is divided into components $C_1^1 = \{\mathtt{E}\}$ and $C_2^1 = \{\mathtt{A},\mathtt{D},\mathtt{C},\mathtt{I}\}$, so $|C_2^1| > 3$, violating the problem constraint. Then, S_1 must be repaired by inserting new nodes into the solution until it becomes feasible. Inserting node *A* into the separator generates solution S_2, which is a feasible solution. After repairing a solution, we may have introduced redundant nodes in the separator that are not required anymore. Therefore, we scan the separator in order to remove those nodes that are not required to maintain the feasibility of the solution. In this case, nodes **B**, **F**, **G** and **H** are removed from the solution, generating solution S_3, which is feasible. In addition, the number of nodes in S_3 is 2, being the best solution found in the path so far.

Once reaching a feasible solution, the path continues with the original unfeasible one, which is S_1. The next solution, S_4, is generated from the move $Swap(S_1, \mathtt{G}, \mathtt{C})$, which is already a feasible solution, disconnecting the network into $C_1^4 = \{\mathtt{E}\}$, $C_2^4 = \{\mathtt{G}\}$ and $C_3^4 = \{\mathtt{A},\mathtt{D},\mathtt{I}\}$, all of them with a number of nodes lesser than or equal to 3. This move may produce a solution with redundant nodes again. In this case, node **F** is not necessary anymore in the solution, so we remove it to generate S_5, obtaining a better solution than S_4.

The last movement, $Swap(S_4, \mathtt{H}, \mathtt{A})$, leads us to reaching S_g, concluding the trajectory between S_i and S_g. At this point, the algorithm returns the best solution found in the trajectory, which is S_3, as it only contains two nodes in the separator. This new solution achieves an improvement of two nodes with respect to the solutions considered to be endpoints of the trajectory.

The choice of the next *Swap* movement to be performed in each iteration results in two different PR strategies. On the one hand, *Greedy PR* (GPR) selects, in each iteration, the best *Swap* movement available. In particular, it generates all the possible solutions that can be generated in the current trajectory, also applying the repair operator. Then, the trajectory continues through the best intermediate solution. On the other hand, *Random PR* (RPR) generates the next solution of the trajectory by randomly selecting one of the available *Swap* moves in the path. Notice that GPR is focused on intensification, sacrificing computational time, while RPR aims for diversification, resulting in the fastest strategy as it does not require a complete analysis of the available *Swap* moves to be performed.

In this work, we propose a third variant that mixes the GPR and RPR, named *Greedy Randomized PR* (GRPR). This new strategy generates all the possible intermediate solutions as in GPR, but instead of selecting the best one, it selects one of the most promising ones, following the same strategy than the GRASP constructive stage. We consider a parameter $\sigma \in [0, 1]$ (equivalent to β in GRASP), which is able to establish a trade-off between greediness and randomness of the strategy. This strategy allows us to create a path focused on intensification but escaping from local optima derived from choosing a totally greedy variant, balancing the diversification and intensification of the search. Notice that evaluating all possible paths between solutions can eventually lead to the evaluation of a large number of intermediate solutions. The number of intermediate solutions is limited by the number of different critical nodes in the solutions considered to construct the path. However, high-quality solutions share a reasonable percentage of critical nodes, so the number of different critical nodes in two high-quality solutions is small, thus leading to a small number of intermediate solutions in the path between them. This behaviour allows the algorithm to evaluate all possible paths between two solutions.

The intermediate solutions generated during trajectory exploration are not necessarily locally optimal with respect to any neighbourhood, even more considering the repair operator. Therefore, the local search method described in Section 2.3 is applied to the best solution found in the path to further improve its quality.

This work considers that all the constructed solutions are included in the ES (avoiding repeated solutions). Hence, PR is applied to every pair of different solutions generated in the constructive and improvement phases of the algorithm.

Finally, a new PR strategy, named *Exterior PR*, is presented. The aforementioned variants (GPR, RPR and GRPR) are focused on constructing a path that connects two solutions. However, if the solutions considered for the path are very similar, then the trajectory will be short, which is not interesting in the PR methodology. The *Exterior PR* (EPR) variant was originally proposed by Duarte et al. (2015) with the aim of exploring trajectories that go beyond the considered solutions. In particular, starting from the initial (S_i) and guiding (S_g) solutions, EPR tries to generate, in each iteration, a solution obtained by removing from the initial solution those elements that are already present in the guiding one. Therefore, the main idea of EPR is not getting closer to the guiding solution but getting further from it. Hence, this strategy is totally focused on the diversification of the search, trying to generate intermediate solutions that are rather different from the two endpoints of the path. EPR ends when the initial solution becomes completely different to the guiding one (i.e. $S_i \cap S_g = \emptyset$), returning the best solution found in the path. Again, the local search method is applied to the best intermediate solution in order to obtain a local optimum.

4 | COMPUTATIONAL RESULTS

This section has two main objectives: 1) find the best values for the parameters of the proposed algorithm and 2) compare the selected variant with the best previous method found in the state of the art. All the algorithms have been developed in Java 9, and the experiments have been conducted in an Intel Core 2 Duo 2.66 GHz with 4GB of RAM computer.

It is important to remark that the instance set used in the best previous method is not publicly available, and we have not received a response from the previous authors regarding this issue. Notwithstanding, the instance set used in this experimentation has been generated using the same graph generator proposed by the best previous work (Lee et al., 2017). Specifically, graphs are generated using the Erdös Rényi model (Erdös & Rényi, 1960), in which each new inserted node has the same probability of being connected to any existent node in the graph. A set of 50 instances (25 with 100 vertices and 25 with 200 vertices) with different density (from 200 to 500 edges) has been generated. Each instance has been tested considering α = {0.2, 0.4, 0.6}. In order to facilitate future comparisons, the instances can be requested by email to the authors.

Experimentation is divided into two different parts: preliminary experimentation and final experimentation. The former is designed to adjust the parameters for the algorithms and select the best version of them, while the latter has the aim of confirming the quality of the proposal by comparing it with the best previous method found in the state of the art. It is worth mentioning that the preliminary experimentation will consider a representative subset of 20 of 50 instances to avoid overfitting.

All the experiments report the following metrics: *Avg*., the average objective function value; *Time (s)*, the average computing time measured in seconds; *Dev*. (%), the average deviation with respect to the best solution found in the experiment; and *#Best*, the number of times the algorithm reaches that best solution.

The first preliminary experiment is designed for analysing the performance of the proposed constructive procedures and selecting the best variant. In particular, this experiment analyses the effect of the β parameter in both constructive procedures presented in Section 2.1. The behaviour of the constructive procedure isolated is not a faithful representation of its adequacy to the problem. This is mainly because certain variants may produce worse quality solutions that are a better starting point for the local improvement method. Therefore, the experiments designed to select the best constructive procedure consists of generating 10 different solutions, improving them with the local search presented in Section 2.3. Considering the value for the β parameter, we have tested β = {0.25, 0.50, 0.75, *RND*}, where RND indicates that a random value is selected in each independent construction. Table 1 shows the results obtained by the *GreedyRandom* constructive procedure.

As it can be derived from these results, increasing the value of β worsens the quality of the solutions obtained, which is reflected in all the considered metrics. This result suggests that the best solutions are reached when reducing the randomness in the constructive procedure.

TABLE 1 Results for the constructive procedure *GreedyRandom* when coupled with the local search method

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GreedyRandom (0.25)	68.35	16.69	1.16	11
GreedyRandom (0.50)	68.95	16.92	2.33	7
GreedyRandom (0.75)	71.35	18.30	6.69	1
GreedyRandom (RND)	68.70	17.61	2.10	10

Note: The number between parenthesis indicates the value of β.

Therefore, we select β = 0.25 for the *GreedyRandom* procedure. Analysing the computing time, we can clearly see that the differences among the considered variants are negligible.

We perform an equivalent experiment for the second proposed constructive method. Specifically, Table 2 shows the corresponding results when considering the *RandomGreedy* procedure.

These results confirm the hypothesis, suggesting that the lower the randomness of the constructive procedure, the better the results. Again, selecting β = 0.25 leads the constructive procedure to obtain the best solutions in all the considered metrics: a larger number of best solutions found while maintaining the smallest deviation with respect to the best solution when it is not reached. Therefore, we again select β = 0.25 for the *RandomGreedy* variant. Once again, the computing time required by each algorithm is rather similar.

To conclude with the constructive procedure selection, we directly compare the results obtained with the best *GreedyRandom* variant with those obtained by the best *RandomGreedy* variant. Table 3 shows the comparison between *GreedyRandom* and *RandomGreedy* when considering α = 0.25.

Analysing the results obtained, the *RandomGreedy* variant is clearly better than the *GreedyRandom* procedure. In particular, it is able to reach a larger number of best solutions (13 versus 15), with a deviation of just 0.77%, while *GreedyRandom* obtains 3.27%. These results suggest that *RandomGreedy* is able to reach most of the best solutions, and in those cases in which it does not reach the best solution, it remains close to the best solution in terms of quality. In this case, the computing time required by the *RandomGreedy* variant is also smaller as it does the evaluation of all the candidate nodes to be included in the solution under construction is not precise.

Furthermore, we can confirm the hypothesis stated in Section 2.1, which indicates that *RandomGreedy* should be faster than *GreedyRandom* as it does not require the complete evaluation of the *CL*. Analysing the results presented in Tables 1, 2 and 3, we can clearly see that *RandomGreedy* is consistently faster than *GreedyRandom* in all their variants. As a result, we select *RandomGreedy* with β = 0.25 as the constructive procedure for the remainder of the paper.

The next experiment aims to analyse the behaviour of the local search when varying the portion of the search space explored. It is well known that an exhaustive search is rather time consuming, thus affecting the efficiency of the complete algorithm. In order to avoid this, we will now analyse two factors, objective function value and computing time, when considering a reduction of the search space explored by the local search method. In order to do so, we test the efficacy and efficiency when exploring a percentage δ of the available movements inside local search. Specifically, we test δ = {0.10, 0.20, ..., 0.90}, with delta being the percentage of nodes considered to be evaluated with the *Move2x1* operator, defined in Section 2.3. It is expected that small values of δ will result in shorter computing times, but it might deteriorate the quality of the solutions found. Figure 3 shows the comparison between quality and computing time when considering different values for the δ parameter.

As can be derived from the results, the search is consistently improving the results without considerably increasing the computing time until reaching δ = 0.6. At that point, the deviation decreases rather slowly (from approximately 4% until reaching 0.5%), while the computing time is rather larger (from about 150 seconds to more than 500 seconds). Hence, considering values larger than δ = 0.6 is not recommended for the local search method. The remaining experiment will then consider δ = 0.6.

Once the best constructive and local search methods have been identified, the next step consists of selecting the best PR variant to be considered. In particular, we propose four different variants: GPR, RPR, GRPR and EPR. Among all the variants, GRPR is the only one with a parameter, σ, that must be adjusted, controlling the balance between intensification and diversification in the combination. Therefore, the next preliminary experiment is devoted to selecting the best value for the σ parameter. In particular, we have tested σ = {0.25, 0.50, 0.75, *RND*} as in the previous experiments. We have generated and improved 10 different solutions with the selected constructive and local search methods and then combined each pair of solutions with the corresponding combination method. Table 4 shows the results derived from this experiment.

Although there are no significant differences in the obtained results, selecting a random value of σ in each iteration reaches slightly better solutions, and the computing time is not drastically affected. Therefore, we select σ = RND for the remaining experiments. Having selected the

TABLE 2 Results for the constructive procedure *RandomGreedy* when coupled with the local search method

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
RandomGreedy (0.25)	67.65	14.75	1.24	11
RandomGreedy (0.50)	67.90	15.46	2.10	8
RandomGreedy (0.75)	68.15	14.82	2.54	3
RandomGreedy (RND)	67.85	14.84	2.25	6

Note: The number between parenthesis indicates the value of β.

TABLE 3 Comparison between best *GreedyRandom* and *RandomGreedy* constructive procedures

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GreedyRandom (α = 0.25)	68.35	16.69	3.27	13
RandomGreedy (α = 0.25)	67.65	14.75	0.77	15

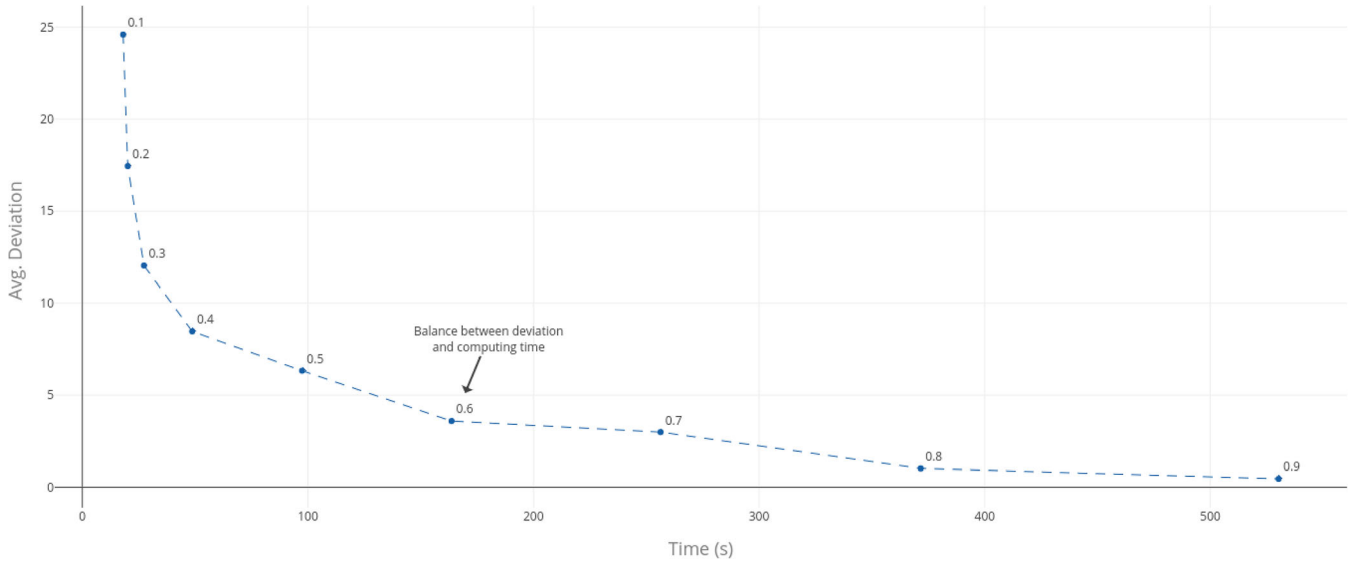

FIGURE 3 Analysis of the average deviation and computing time of each variant of the local search procedure

TABLE 4 Adjustment of the σ parameter in the Greedy Randomized Path Relinking algorithm

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GRPR (0.25)	61.40	825.10	0.53	16
GRPR (0.50)	61.45	837.83	0.59	15
GRPR (0.75)	61.45	833.52	0.59	15
GRPR (RND)	61.25	835.42	0.49	16

Note: The number between parenthesis indicates the value of σ.
Abbreviations: GRPR, Greedy Randomized Path Relinking.

TABLE 5 Comparison among the proposed Path Relinking variants

Algorithm	Avg.	Time (s)	Dev. (%)	#best
GRPR (RND)	61.25	835.42	0.89	13
RPR	61.25	838.58	0.91	13
GPR	61.25	839.07	0.96	13
EPR	61.20	845.52	0.85	13

Abbreviations: EPR, Exterior Path Relinking; GPR, Greedy Path Relinking; GRPR, Greedy Randomized Path Relinking; RPR, Random Path Relinking.

best GRPR variant, we now perform a comparison among all the PR variants with the aim of selecting the best one for the final experiment. Table 5 shows the comparison among RPR, GPR, GRPR and EPR.

As can be derived from the presented results, the values obtained by the different variants are rather similar. In particular, each variant is able to obtain 13 of 20 best solutions in similar computing times. We then select EPR as the best variant as it presents the smallest deviation (0.85%), although it is closely followed by GRPR (0.89%). The superiority of EPR can be partially justified as it is totally focused on the diversification of the search. If we analyse the constructive and local search methods, we can clearly see that they are mainly focused on the intensification part of the search, so the combination with EPR generates an adequate balance between diversification and intensification. Regarding the computing time, although EPR presents a slightly larger computing time, the quality of the solutions obtained makes up for it. Then, we select EPR as the best PR variant for the α-SP.

The final experiment selects the best variant and compares it with the best previous method found in the state of the art. In particular, the proposed GRASP with PR (GRASP+PR) algorithm considers the following components:

- Constructive Procedure: *RandomGreedy*(0.25)
- Local Search: δ = 0.6
- PR: EPR

As far as we know, the best previous algorithm for the α-SP is a Random Walk algorithm (RW) based on a Metropolis chain (Lee et al., 2017). Unfortunately, neither the original source code nor the original instances are available, so we have carefully re-implemented the original algorithm following the detailed descriptions given in the original work. It is important to remark that this experiment considers the complete set of 50 instances. In addition to GRASP+PR, we have also included in this final comparison the results obtained by GRASP (without applying the PR algorithm) as, in some scenarios, it is more important to provide a fast solution of average quality than to spend several seconds trying to refine the best solution found. Table 6 shows the results obtained in the comparison.

Regarding these results, the proposed algorithm, GRASP+PR, is able to reach a larger number of best solutions than RW (34 vs 18). Furthermore, the average deviation with respect to the best solution found is considerably small in GRASP+PR, indicating that, in those cases in which it is not able to reach the best solution, it remains rather close to it. On the contrary, the deviation of RW is 18.26%, which means that it is not close to the best solution found by GRASP+PR. Analysing the computing time, we can see that GRASP+PR is 1.30 times faster than RW, being more adequate for real scenarios. Finally, RW is able to disconnect the network by removing, on average, 71.78 nodes, while GRASP+PR can disconnect it by removing just 62.00 nodes on average, which also confirms the quality of the proposal.

In order to confirm that there are statistically significant differences between the compared algorithms, thus confirming the superiority of the proposal, we perform the pairwise Wilcoxon Signed Rank Test. The resulting p-value of .001 indicates that there are statistically significant differences between GRASP+PR and RW with a significance level of 95%. Therefore, GRASP+PR emerges as the best algorithm in the state of the art for the α-SP.

As has been mentioned throughout the manuscript, computing time is a key factor in the context of α-SP. In order to analyse this relevant issue, we conducted an additional experiment to compare the computing time required by both, the proposed algorithm, GRASP+PR, and the best method identified in the literature, RW. Figure 4 shows the computing time required by each algorithm in every single instance considered in the experimentation. As we can see, GRASP+PR is clearly faster than RW. However, an average computing time of approximately 800 seconds may not be enough in some real-case scenarios.

In view of the results obtained, we can conclude that GRASP+PR and RW require equivalent computing times when facing large instances ($n \geq 200$), although GRASP+PR is considerably faster in small and medium instances ($n < 200$). Therefore, if a fast solution is needed in special applications, we do recommend considering GRASP without applying PR, which is able to obtain a slightly larger deviation than GRASP+PR (5.90

TABLE 6 Final comparison between GRASP, GRASP + PR and the best previous method found in the state of the art

Algorithm	Avg.	Time (s)	Dev. (%)	#best
GRASP	63.18	137.41	5.90	14
GRASP + PR	62.00	822.29	3.68	34
RW	71.78	1070.36	18.26	18

Abbreviations: GRASP, Greedy Randomized Adaptive Search Procedure; PR, Path Relinking; RW, Random Walk algorithm.

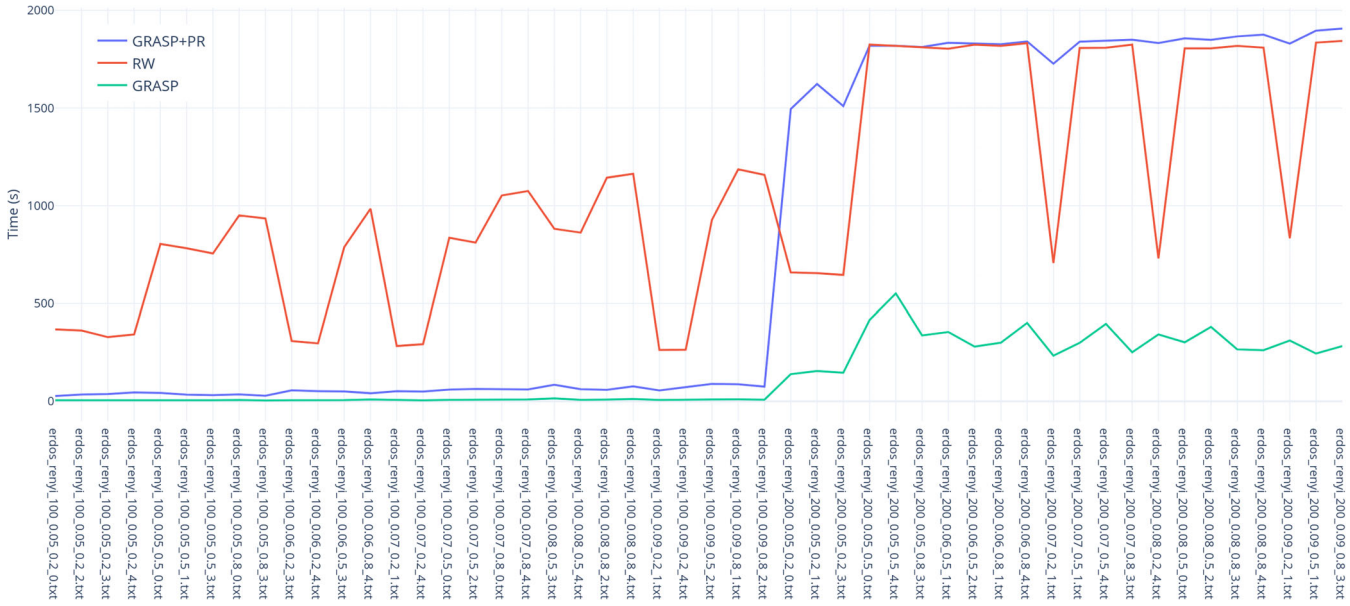

FIGURE 4 Comparison of computing time for each one of the considered instances

vs 3.68%) but requires an eighth of the time (137.41 s) needed by GRASP+PR or RW (822.29 s and 1070.36 s, respectively). Furthermore, analysing the graph, we can see that GRASP scales better with the size of the instance.

5 | CONCLUSIONS

In this work, an algorithm based on GRASP has been presented for the detection of critical points in networks, particularly a constructive procedure that leverages features derived from SNA, coupled with an improvement strategy for reaching local optima. In addition, the generated solutions are combined by using PR to improve their quality. We perform a thorough experimentation for adjusting the parameters of the algorithm and show the merit of each proposed strategy. Finally, the best variant is compared with the best algorithm found in the state of the art. As can be derived from the experiments, the proposed algorithm is able to solve the α-SP, being 1.30 faster than the previous method, obtaining clearly better results, which are supported by non-parametric statistical tests. The proposed algorithm emerges as the best method in the state of the art, being able to obtain high-quality solutions in short computing times with GRASP and improve them later with PR in those situations in which time is not critical.

ACKNOWLEDGEMENTS

This work has been partially funded by 'Ministerio de Ciencia, Innovación y Universidades' under grant ref. PGC2018-095322-B-C22 and by 'Comunidad de Madrid' and 'Fondos Estructurales' of European Union with the project 'Cybersecurity, Network Analysis and Monitoring for the Next Generation Internet' (CYNAMON), with grant ref. S2018/TCS-4566.

CONFLICT OF INTEREST

The authors confirm that there are no conflicts of interest.

ORCID

Sergio Pérez-Peló https://orcid.org/0000-0002-1915-4160
Jesús Sánchez-Oro https://orcid.org/0000-0003-1702-4941
Abraham Duarte https://orcid.org/0000-0002-4532-3124

ENDNOTE

[1] An Autonomous System conforms to one or more networks managed and supervised by a single entity or organization.

REFERENCES

Andersson, G., Donalek, P., Farmer, R., Hatziargyriou, N., Kamwa, I., Kundur, P., … Vittal, V. (2005). Causes of the 2003 major grid blackouts in North America and Europe, and recommended means to improve system dynamic performance. *IEEE Transactions on Power Systems*, *20*(4), 1922–1928.

Ben-Ameur, W., Mohamed-Sidi, M.-A., & Neto, J. (2015). The k-separator problem: Polyhedra, complexity and approximation results. *Journal of Combinatorial Optimization*, *29*(1), 276–307.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, *25*(2), 163–177.

Crucitti, P., Latora, V., & Marchiori, M. (2004). Model for cascading failures in complex networks. *Physical Review E*, *69*, 045104.

Duarte, A., Sánchez-Oro, J., Resende, M. G. C., Glover, F., & Martí, R. (2015). Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences*, *296*, 46–60.

Erdös, P., & Rényi, A. (1960). On the evolution of random graphs. *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, *5*(1), 17–60.

Feige, U., & Mahdian, M. (2006). Finding small balanced separators. In Proceedings of the thirty-eighth annual ACM symposium.

Feo, T. A., & Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67–71.

Feo, T. A., Resende, M. G. C., & Smith, S. H. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, *42*(5), 860–878.

Garey, M., & Johnson, D. (1979). *Computers and intractability - A guide to the theory of NP-completeness*. San Francisco, CA: Freeman.

Glover, F. (1996). *Tabu search and adaptive memory programming – Advances, applications, and challenges*. Dordrecht, MA: Kluwer Academic Publishers.

Glover, F., & Laguna, M. (1998). Tabu search. In *Handbook of combinatorial optimization* (pp. 2093-2229). Boston, MA: Springer.

Glover, F., Laguna, M., & Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, *29*(3), 653–684.

Glover, F. W., & Kochenberger, G. A. (Eds.). (2006). Handbook of metaheuristics (Vol. 57). Springer Science & Business Media.

Hansen, P., & Mladenović, N. (2014). Variable neighborhood search. In *Search methodologies* (pp. 313–337). Boston, MA: Springer.

Laguna, M., & Martí, R. (1999). Grasp and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, *11*(1), 44–52.

Lee, J., Kwak, J., Lee, H. W., & Shroff, N. B. (2017, March). *Finding minimum node separators: A markov chain monte carlo method*. Paper presented at the Drcn 2017 - Design of Reliable Communication Networks; 13th International Conference (pp. 1–8).

Li, R., Hu, S., Zhang, H., & Yin, M. (2016). An efficient local search framework for the minimum weighted vertex cover problem. *Information Sciences*, *372*, 428–445.

Mohamed-Sidi, M. (2014). *K-separator problem (Problème de k-Sèparateur)*. (Unpublished doctoral dissertation). Telecom & Management SudParis, Èvry, Essonne, France.
Newman M. E. J. (2008) Mathematics of Networks. In P. Macmillan, (Ed) *The New Palgrave Dictionary of Economics*. Palgrave Macmillan, London.
Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). *The pagerank citation ranking: Bringing order to the web* (1999-66). Stanford InfoLab.
Quintana, J. D., Sánchez-Oro, J., & Duarte, A. (2016). Efficient greedy randomized adaptive search procedure for the generalized regenerator location problem. *International Journal of Computational Intelligence Systems*, *9*(6), 1016–1027.
Sánchez-Oro, J., Laguna, M., Martí, R., & Duarte, A. (2016). Scatter search for the bandpass problem. *Journal of Global Optimization*, *66*(4), 769–790.
Scott, J., & Carrington, P. J. (2011). The *SAGE handbook of social network analysis*. SAGE publications.
Wachs, M., Grothoff, C., & Thurimella, R. (2012). Partitioning the internet. In 2012 7th International Conference on Risks and Security of Internet and Systems (CRiSIS) (pp. 1-8). IEEE.
Yannakakis, M. (1981). Node-deletion problems on bipartite graphs. *SIAM Journal on Computing*, *10*(2), 310–327.
Yen, J. Y. (1971). Finding the k shortest loopless paths in a network. *Management Science*, *17*(11), 712–716.

AUTHOR BIOGRAPHIES

Sergio Pérez-Peló was born in Madrid (Spain) on June 21, 1995. He has achieved his double degree in Computer Sciences and Software engineering from the Universidad Rey Juan Carlos (URJC) in 2017 and his M.S in Cibersecurity in 2018 from the Universidad Oberta de Cataluña (UOC). He is actually working in his PhD in URJC as a member of the Group for Research on Algorithms For Optimization (GRAFO). His main research interests focus on Artificial Intelligence and Operations Research, focusing in Social Network Analysis and real-world optimization problems.

Jesús Sánchez-Oro was born in Madrid (Spain) on December 31, 1987. He holds a degree in Computer Science from the Universidad Rey Juan Carlos (2010), his Master's degree in Computer Vision from the Universidad Rey Juan Carlos in 2011, and his Ph.D. in Computer Science in 2016 from the Universidad Rey Juan Carlos. He is visiting professor at the Computer Science Department, and he is a member of the Group for Research on Algorithms For Optimization (GRAFO). His main research interests focus on Artificial Intelligence and Operations Research, specially in heuristics and metaheuristics for solving hard optimization problems.

Abraham Duarte was born in Hervás (Cáceres, Spain) on October 24, 1975. He received his M.S. degree in Physics Sciences (Electronic Speciality) from the Universidad Complutense de Madrid (UCM) in 1998 and his Ph.D. in Computer Science in 2004 from the Universidad Rey Juan Carlos (URJC). He is Full Professor at the Computer Science Department where is the leader of the Group for Research on Algorithms For Optimization (GRAFO). His main research interests focus on the interface among Computer Science, Artificial Intelligence and Operations Research. Most of his publications deal with the development of metaheuristics procedures for optimization problems.

How to cite this article: Pérez-Peló S, Sánchez-Oro J, Duarte A. Finding weaknesses in networks using Greedy Randomized Adaptive Search Procedure and Path Relinking. *Expert Systems*. 2020;e12540. https://doi.org/10.1111/exsy.12540