Received: 12 November 2021 | Revised: 25 October 2022 | Accepted: 8 November 2022

DOI: 10.1002/net.22134

RESEARCH ARTICLE

WILEY

Variable neighborhood search approach with intensified shake for monitor placement

Alejandra Casado[1] | **Nenad Mladenović**[2] | **Jesús Sánchez-Oro**[1] | **Abraham Duarte**[1]

[1]Department of Computer Sciences, Universidad Rey Juan Carlos, Móstoles, Spain

[2]Department of Industrial Engineering and Research Center on Digital Supply Chain and Operations Management, Khalifa University, Abu Dhabi, UAE

Correspondence

Abraham Duarte, Department of Computer Sciences, Universidad Rey Juan Carlos, Móstoles, Spain.

Email: abraham.duarte@urjc.es

Funding information

Ministerio de Ciencia, Innovación y Universidades, Grant/Award Number: PGC2018-095322-B-C22; Comunidad de Madrid, Grant/Award Number: S2018/TCS-4566; Fondos Estructurales, Grant/Award Number: Y2018/EMT-5062; Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan, Grant/Award Number: AP08856034.

Abstract

Several problems are emerging in the context of communication networks and most of them must be solved in reduced computing time since they affect to critical tasks. In this research, the monitor placement problem is tackled. This problem tries to cover the communications of an entire network by locating a monitor in specific nodes of the network, in such a way that every link remains surveyed. In case that a solution cannot be generated in the allowed computing time, a penalty will be assumed for each link uncovered. The problem is addressed by considering the variable neighborhood search framework, proposing a novel constructive method, an intelligent local search to optimize the improvement phase, and an intensified shake to guide the search to more promising solutions. The proposed algorithm is compared with a hybrid search evolutionary algorithm over a set of instances derived from real-life networks to prove its performance.

KEYWORDS

intensified shake, metaheuristics, monitor placement, variable neighborhood search

1 | INTRODUCTION

Communication networks are, nowadays, an essential part of every activity in both personal and professional points of view. Among their services we can highlight streaming, security, banking, shopping, and so forth. Originally, the security of those networks was not a requirement when designing them. However, the increase in the network attacks has made security a key factor in the network design of every company.

Although a successful attack to the network of a company may result in an important economic crisis, it must be noted that there are some critical services that must be continuously monitored, such as transport, hospitals, or defense. The failure in these kinds of services usually results in severe damages to people which may result in humanitarian crisis [1].

The number of attacks is continuously increasing, as well as the privacy on the Internet is becoming more and more relevant every year. As a result, companies and institutions need to improve the security system of their networks in order to minimize the possibilities of being attacked, thus increasing the network protection. It is worth mentioning that a fast reaction to an attack is a key factor in successfully protecting it [14].

One of the most common attacks is denial of service (DoS) and distributed denial of service (DDoS), since a successful attack can completely disable an Internet provider. The failure become critical if more services directly depend on the services under attack, which can result in a cascade failure, harming a large number of clients [6].

The early detection of potential threats such as malware spreading, loss of certain nodes in the network, unauthorized accesses to the network, among others, has a positive impact not only in the security, but also in the network performance. Therefore, it is desirable to perform detection of failures in real time to mitigate the negative effect of the attack.

One of the most important stages when increasing network protection consists of guaranteeing the surveillance of the complete network. If the network is correctly monitored, the system will be able to provide a faster reaction to an attack. This surveillance is commonly performed by deploying a monitor, which is a special hardware, in certain nodes of the network to analyze all the communications that go through them. This analysis provides the network administrators with real-time information about possible threats, which may help them to protect specific nodes.

The most simple approach to perform a complete network analysis consists of deploying a monitor in every node of the network, guaranteeing that all the communications going through the network are captured. Unfortunately, this is not possible since the cost of deploying a monitor in a node is usually high. Additionally, deploying a monitor implies an overhead in the network performance since all the traffic must be captured. Therefore, the number of monitors deployed in a network must be minimized while maintaining the complete network analyzed.

The monitor placement problem (MPP) has been widely studied in the literature, resulting in different variants which consider new aspects of the network surveillance. Although traditional approaches consider static networks, where changes are not common, in this work we are focused in networks which are in constant evolution, so the deploying of monitors must be optimized to achieve real-time analysis of the network without negatively affecting to its performance.

This problem can be reduced to a vertex covering problem, which is $\mathcal{NP}$-complete [13] and, therefore, exact approaches are not able to solve real-life instances due to their complexity. The problem remains $\mathcal{NP}$-complete for approximations within a factor smaller than 1.36 [7].

Therefore, this family of problems has been mainly tackled from a heuristic perspective. First approaches were focused on evolutionary algorithms for the maximum independent set [2] and the minimum vertex cover [9]. Later, a branch and bound and different heuristics were proposed for solving the minimum vertex cover when considering random graphs [15,16]. More approaches were then proposed for the same problem: genetic algorithms [12], simulated annealing [11], or a hierarchical Bayesian algorithm [22], among others. Milanovic [17] proposed a new genetic algorithm for solving the generalized vertex cover problem considering networks with weights in both, nodes and links. More recently, Chandu [5] designed a parallel evolutionary algorithm that leverages the hardware architecture by considering Apache Hadoop for distributing computation.

Evolutionary algorithms have been widely studied in the field of monitor placement, from a population injection method [18] to a hybrid evolutionary algorithm devoted to solve a dynamic variant of the problem [19]. As far as we know, the best algorithm in the literature for the MPP is a hybrid search heuristic [20]. This procedure is an enhancement of the population injection method originally presented in [18], whose main contribution is the increase in the diversity of the search.

In this work, we address the MPP with a variable neighborhood search (VNS) algorithm, which consists of a novel constructive method, an intelligent local search to optimize the improvement phase, and an intensified shake to guide the search to more promising solutions. The remaining of the paper is structured as follows. Section 2 formally describes the problem tackled in this research, Section 3 presents the algorithmic proposal to solve the MPP, Section 4 shows the experimental analysis performed to validate the proposal, comparing it with a hybrid search evolutionary algorithm and, finally, Section 5 draws some conclusions derived from this research.

2 | PROBLEM DEFINITION

The objective of the MPP is to monitor a complete network, which is usually modeled as an undirected graph $G = (V, E)$, where V, with $|V| = n$, is the set of nodes in the network, and E, with $|E| = m$, consists of pairs of nodes (u, v), with $u, v \in V$, indicating that there is a connection between nodes u and v.

Before defining the problem under consideration, it is necessary to introduce the concept of vertex cover. Specifically, a vertex cover Λ of a network is a subset of vertices $\Lambda \subseteq V$ which satisfies the constraint that, for every edge $(u, v) \in E$, either $u \in \Lambda$ or $v \in \Lambda$. Then, the minimal vertex cover is the one with the minimum size. More formally, the minimum vertex cover problem is defined as follows:

$$MVCP(G) = \min_{\Lambda \in C} |\Lambda|,$$

where C is the set of all possible vertex covers for the network G. Since a network can have several different minimum vertex covers (naturally, with the same value of objective function), we refer to the MVCP as the problem of finding one of those vertex covers.

The MPP is a particular variant of the MVCP but with direct application in communication networks. In this context, there are some additional features that must be satisfied in order to solve the MPP. The first one is related to the computing time required to solve the problem. In real-life networks, the time window available to solve the MPP is considerably smaller than in the classic MVCP. The rationale behind this consideration is the immediacy with which any anomalous network behavior must be solved, with the aim of activating or deactivating any node producing the anomaly in the monitored network. In case that the time feature cannot be satisfied then, the budgeted time should be used for finding a solution which monitors the largest number of connections in the network. In this case, not all the connections are equally relevant, so it is necessary to prioritize the links to be monitored [3].

A network is considered to be fully monitored if and only if all the links are covered by, at least, one monitor. The optimal solution is the one that covers the complete network while minimizing the impact in the performance. Therefore, the MPP looks for monitoring the complete network but placing the minimum number of monitors in it. For the sake of simplicity, we consider that a monitor can be placed in any node of the network.

The definition of priority for a link in the network completely depends on its nature. For instance, it can be related to the traffic flow through the link, its bandwidth, the relevance of the link in the real network, among others. We refer the reader to [21,25] for a detailed analysis of setting network priorities.

The inclusion of link priorities prevents the MPP to be reformulated as the MVCP. However, as stated in [20], the priorities can be easily included in the problem model by considering a priority function $p : E \rightarrow \mathbb{N}$ that assigns a penalty to each uncovered link in a given solution. Therefore, two solutions with the same number of monitors can be compared by computing the penalty due to the uncovered links.

The objective function for the MPP is then conformed by two different elements: the number of monitors that are deployed, and the accumulated penalty due to the uncovered links. A solution S (with $S \subseteq V$) for the MPP contains the nodes in which a monitor should be deployed. Given a solution S, the objective function of the MPP is evaluated as:

$$MPP(S) = |S| + \sum_{(u,v)\in E'} p(u,v),$$

where $E' = \{(u,v) \in E : u \notin S \wedge v \notin S\}$, that is, the set of all edges of the network which are not covered by any monitor. Then, the objective of the MPP is to find the solution $S^\star$ with the minimum value of the objective function. More formally,

$$S^\star = \arg\min_{S\in\mathbb{S}} MPP(S),$$

where $\mathbb{S}$ represents the solution space of the problem, consisting in all the feasible solutions for the MPP. It is worth mentioning that any subset of nodes conforms a feasible solution. In particular, $|S| = |V|$ means that a monitor is deployed in every single node. Then, the first addend takes on the largest value while the second addend is zero. Similarly, $|S| = 0$ means that no monitor is deployed. Therefore, the first addend is zero, while the second one acquires its maximum value.

The objective function balances both features: the number of monitors included in the solution and the penalties associated with those links which are not covered. Although there are several types of penalties that can be considered for the network, we follow the same approach as in [20], which considers a static linear distance penalty function (see Section 4 for the penalty included in the considered instances).

Figure 1 shows a network instance consisting of 5 nodes, $V = \{1,2,3,4,5\}$ and 5 edges, $E = \{(1,2),(1,4),(2,3),(2,5),(3,5)\}$. The number close to each edge indicates the penalty assumed if this link is not covered. Figure 2 represents three different solutions for Figure 1. In all of them, the nodes where a monitor is deployed are colored, while the links which are

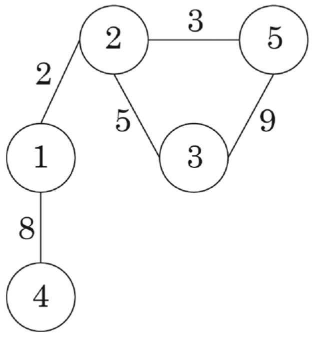

FIGURE 1 Example instance with 5 nodes and 5 edges

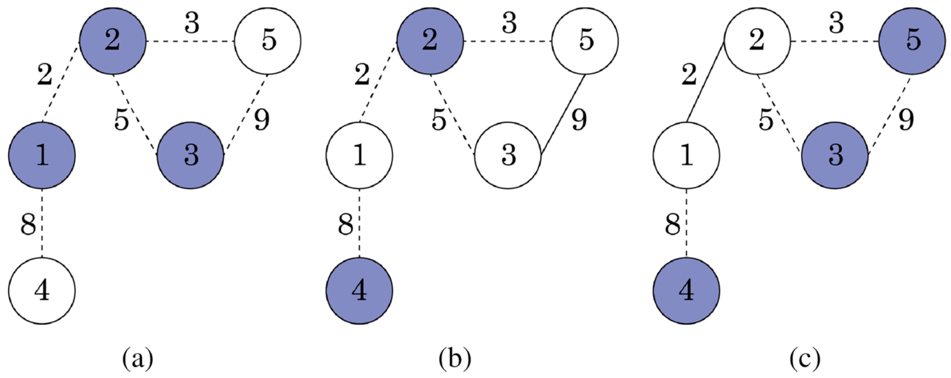

FIGURE 2 Three possible solutions for the instance depicted in Figure 1. (A) Solution $S_1 = \{1, 2, 3\}$, (B) solution $S_2 = \{2, 4\}$, and (C) solution $S_3 = \{3, 4, 5\}$

covered by, at least, one monitor, are represented with a dashed line. Figure 2A depicts solution S_1, selecting nodes 1, 2, and 3, which are able to cover the entire network. Therefore, the objective function value is evaluated as $MPP(S_1) = 3 + 0 = 3$, since there are 3 nodes in the solution and there is no penalty (all links are covered). If we now analyze Figure 2B, there are two selected nodes (2 and 4) instead of three, but the link $(3, 5)$ is not covered by any deployed monitor. Then, the objective function value is $MPP(S_2) = 2 + 9 = 11$. Finally, solution S_3 deploys a monitor in nodes 3, 4, and 5, but the link (1,2) is still uncovered, resulting in an objective function value of $MPP(S_3) = 3 + 2 = 5$. Regarding these results, the best solution is S_1, which provides the smallest objective function value. It is interesting to analyze that S_3 outperforms S_2 even having selected more nodes to deploy a monitor. In the context of MPP, a solution with all the links covered is always better than a solution with at least one link uncovered but, due to time constraints, it is not always possible to cover all the nodes.

3 | ALGORITHMIC APPROACH

In contrast to most of the previous works, which followed approaches based on evolutionary algorithms, this paper tries a novel strategy where a trajectory-based metaheuristic is proposed. While population-based metaheuristics, such as evolutionary algorithms, maintain a population of solutions which are combined to continue improving, trajectory-based metaheuristics work with a single solution which is iteratively improved following different stages.

Specifically, this work is focused on VNS as metaheuristic framework for solving the MPP. The success of VNS relies on systematic changes of neighborhoods with the aim of obtaining high quality solutions in small computing times. In the last decades, VNS has been in continuous evolution, giving rise to a large variety of schemes, which are usually classified depending on the neighborhood exploration. In particular, the neighborhood proposed can be explored by following three different criteria: deterministic (variable neighborhood descent), stochastic (reduced VNS), or mixed (basic VNS). Additional variants have also been presented: General VNS, variable neighborhood decomposition search, variable formulation search, among others. See [10] for further details.

This work considers the basic VNS (BVNS) scheme, which combines the intensification of the local improvement stage with the diversification introduced by the perturbation method. Finding a balance between both, intensification and diversification, will eventually lead to high quality solutions. Algorithm 1 shows the pseudocode of the proposed BVNS.

The method requires three input parameters: the initial solution S (see Section 3.2 for more details), the maximum neighborhood to be explored during the search $k_{\max}$, and the maximum number of iterations performed δ. BVNS is usually executed either a certain number of iterations or for a fixed computing time. In this case, we fix the number of iterations δ (steps 1–13). In each iteration, the algorithm starts with the first neighborhood (step 2). Then, BVNS iterates until reaching the largest predefined neighborhood $k_{\max}$ (steps 3–12). For each neighborhood, the solution is perturbed following the shake procedure described in Section 3.4 (step 4). The perturbed solution S' is then improved using the method described in Section 3.3 to find a local optimum in the current neighborhood (step 5). Finally, BVNS performs the neighborhood change stage. In particular, if the improved solution S'' outperforms the best solution found so far (step 6), the method restarts the search from the first neighborhood (step 7), updating the best solution found (step 8). Otherwise, the algorithm continues with the next neighborhood (step 10). BVNS ends when performing δ iterations, returning the best solution found during the search (step 14).

Algorithm 1. *BVNS*$(S, k_{\max}, \delta)$

```
for $i \in 1 \dots \delta$ do
    $k \leftarrow 1$
    while $k \le k_{\max}$ do
        $S' \leftarrow Shake(S, k)$
        $S'' \leftarrow Improve(S')$
        if $MPP(S'') < MPP(S)$ then
            $k \leftarrow 1$
            $S \leftarrow S''$
        else
            $k \leftarrow k + 1$
        end if
    end while
end for
return $S$
```

3.1 | Support and leaf nodes

Before describing in detail each part of the proposed algorithm, we need to introduce the concept of support nodes, which allows us to reduce the search space. Specifically, there are some nodes that are always in the optimal solution due to the structure of the graph. We denote these nodes as support nodes.

Given a network $G = (V, E)$, we denote $LN \subseteq V$ as the set of leaf nodes which are connected to a single node in the graph (i.e., the degree of the node is equal to 1). The nodes to which the leaf nodes are connected are named as support nodes SN. Considering the example depicted in Figure 1, $LN = \{4\}$ and $SN = \{1\}$, respectively. Then, if we need to monitor all the links in a network, there are only two options to cover the link in which a leaf node is involved: deploy a monitor in the leaf node, or deploy it in its support node. It trivially holds that deploying a monitor in the leaf node always result in an equal or worse solution, since it only covers the link in which the leaf node is involved. Therefore, one optimal solution will always contain a monitor in every support node.

3.2 | Initial solution

VNS requires an initial solution to start the search. This solution can be generated either at random or using a more elaborated procedure. Although the basis of VNS indicates that the initial solution is not relevant for the algorithm, it has been experimentally tested that a high quality initial solution usually helps the algorithm to provide better solutions in smaller computing times [24]. Therefore, in this work, we propose two greedy constructive procedures to provide BVNS a good starting point in the search space.

The first constructive procedure follows a traditional greedy approach which starts from an empty solution and, iteratively, adds elements to it until reaching a feasible solution. Algorithm 2 shows the pseudocode of the proposed constructive procedure.

Algorithm 2. *GreedyConstructive*(G)

```
$S \leftarrow \emptyset$
$DeploySupportNodes(S, G)$
$UL \leftarrow \{(u, v) \in E : u \notin S \wedge v \notin S\}$
$C \leftarrow V \setminus (LN \cup SN)$
while $UL \neq \emptyset$ do
    $c \leftarrow \arg\max_{u \in C} \sum_{(u,v) \in E, v \in C} p(u, v)$
    $S \leftarrow S \cup \{c\}$
    $C \leftarrow C \setminus \{c\}$
    $UL \leftarrow UL \setminus \{(c, x) : \forall x \in N(c)\}$
end while
return $S$
```

The algorithm starts from an empty solution S (step 1) and deploy a monitor in every support node (step 2). Then, the list of uncovered links UL is created with every edge in the graph which is not covered by any monitor (step 3). The candidate nodes to host a monitor are those which are neither a leaf node nor a support node (step 4). The method now iterates until covering all the links in the network (steps 5–10). In each iteration, the next candidate node is selected as the one which is able to minimize the penalty that affects to the solution under construction (step 6). In other words, we only consider (by summing up the corresponding penalty) edges whose both endpoints belong to C.

The selected candidate c is then added to the solution (step 7), updating the set of candidates (step 8), removing from the set of uncovered links UL all edges in which c is involved (step 9). Notice that $N(c)$ refers to the adjacent nodes to c. The method ends returning a solution S where all the links are covered (step 11).

The second constructive method, named *GreedyDestructive*, follows a destructive approach. In particular, if all links need to be covered in a network, it is expected that a large number of nodes would finally be selected. Therefore, it would be interesting to start from a solution with a monitor deployed in every node but in the set of leaf nodes LN and, iteratively, remove a monitor from a node while all the links remain covered. This destructive approach is useful for those hard optimization problems in which the size of the expected solution consists of selecting the majority of available elements, since it requires less iterations than considering a traditional constructive procedure. Therefore, it can be easily adapted to several combinatorial optimization problems where the objective is to select an eventually large subset of elements.

In each iteration, a monitor can be removed from the node v if and only if it is not a support node (i.e., $v \notin SN$) and all its adjacent nodes have a monitor deployed (i.e., $u \in S \ \forall u \in N(v)$). Among all the monitors that can be removed, we need to select the most promising one. Specifically, we select the node with the smallest degree. The rationale behind this is that the smaller the degree, the smaller the number of links covered by a monitor deployed in that node. The method stops when no monitor can be removed without leaving a link uncovered. For the sake of brevity, we omit the inclusion of the corresponding pseudocode since it is symmetrical to the one reported in Algorithm 2.

3.3 | Improvement method

The local improvement phase in VNS is responsible for finding a local optimum with respect to certain neighborhood. It can be as complex as desired: from a straightforward hill-climbing local search method (see [4]) to a more elaborated metaheuristic such as tabu search, or VND, among others (see [8,23] for some successful applications of using complex metaheuristics as improvement). In this research, we propose the use of a simple yet effective local search heuristic to reach a local optimum with a relatively small computational effort, which is a critical part of the problem.

The definition of a local search method requires three main elements: the movement used in the local search, the neighborhood of solutions that can be generated with the movement, and the strategy selected to traverse that neighborhood.

First of all, it is necessary to define the movement considered for this improvement phase. Notice that the goal of this movement is to reach a solution of better quality than the original one, so we need to guarantee that the movement can produce an improvement. In the context of MPP, since the objective function value is computed as the number of deployed monitors plus the penalty, it is desirable to reduce the number of deployed monitors without increasing the penalty. To that end, we propose a movement named *Exchange2x1* which removes two deployed monitors and locate a new one. According with Section 3.1, support nodes are always in the solution, while leaf nodes are never considered in the solution. Therefore, this move can be mathematically described as follows:

$$\textit{Exchange2x1(S,u,v,w)} \leftarrow (S \setminus \{u, v\}) \cup \{w\},$$

where $u, v \in (S \setminus SN)$, $w \notin S$, and $w \in (V \setminus LN)$. In order to simplify the notation, we denote as $\overline{S} = S \setminus SN$ and $\overline{V} = V \setminus LN$.

Having defined the movement used in this phase, the neighborhood of a solution S is conformed with those solutions that can be reached by performing a single *Exchange2x1* move over S. More formally:

$$N_{2x1}(S) \leftarrow \{S' \leftarrow \textit{Exchange2x1}(S, u, v, w) : \forall u, v \in \overline{S} \wedge w \in (\overline{V} \setminus S)\}.$$

With the movement and neighborhood defined, it is now necessary to indicate the strategy used to traverse the neighborhood. We propose two different strategies, resulting in two independent local search methods: an exhaustive local search and an intelligent local search. The main difference between both strategies is that the latter only considers those movements that do not leave any link uncovered.

The exhaustive local search, named ELS, randomly traverses the neighborhood N_{2x1} by removing each pair of nodes and inserting a new one to replace them. Then, if the resulting solution outperforms the original one, the best solution found is updated, resulting in a first improvement strategy. Otherwise, the movement is undone, continuing the search with new candidates. As can be expected, this strategy performs a vast number of operations, resulting in a nonefficient improvemen t phase (see Section 4 for a detailed analysis).

With the aim of providing an efficient strategy to explore the neighborhood, we propose an intelligent local search, named ILS, that carefully selects both, the two nodes to be removed and the one to be included. The goal of this strategy is to reduce the number of operations to reach a local optimum.

We first consider a preprocessing strategy in charge of identifying extra monitors whose removal does not leave any link uncovered and, therefore, they are not necessary in the solution. In general, given a solution S and a node $u \in S$, whose neighborhood $N(u) \subseteq S$, then, u can be simply removed from S.

The next step in the ILS strategy consists of selecting the nodes involved in the movement. The goal is to select only nodes that drive to an improvement in the objective function. In order to do so, the first node to be removed u is selected at random. Thus, it restricts the other two nodes involved in the movement: v, the other node to be removed; and w, the node which will be included in the solution.

We first identify those monitors whose removal do not drive to an improvement. Formally, given a solution S and any node $u \in S$, such that $|N(u) \setminus S| > 1$, independently of the election of a node v, it is impossible to find a single node w able to cover those links previously covered by u and v.

Therefore, given a solution S, a node $u \in S$ is a promising candidate to be removed if $|N(u) \setminus S| = 1$ (notice that the situation $|N(u) \setminus S| = 0$ was considered in the preprocessing strategy). The candidate to be incorporated in the solution is the only node w that remains in the set $N(u) \setminus S$. Finally, this move is actually an improvement move if we are able to identify a node $v \in (S \cap N(w))$ such as $N(v) \setminus S = \{w\}$.

ILS traverses the neighborhood performing just those moves that guarantee an improvement, thus ignoring those movements that lead to nonimproving solutions. This approach is suitable for those optimization problems in which it is possible to rank the moves evaluated in the local search method. The computational results will show the relevance of performing this study on which elements are the most promising ones, becoming a strategy to be taken into account in similar problems.

3.4 | Shake

The shaking phase in VNS is intended to allow the algorithm to escape from local optima when the local search gets stuck. This component introduces diversification in the search by perturbing the incumbent solution, resorting to a different solution in the neighborhood under exploration. To that end, a parameter k controls the perturbation size. This parameter is responsible for increasing the perturbation size when no improvement is found with the aim of exploring further regions of the search space. On the contrary, this method reduces the perturbation size when finding improvements so as to focus on promising regions of the search space.

The perturbation is performed in two stages, by using two new movements: *Drop* and *Add*. The former removes a monitor deployed at node u in solution S producing a new solution S'. In mathematical terms,

$$Drop(S, u) \leftarrow S \setminus \{u\}.$$

On the contrary, the *Add* movement, denoted as $Add(S, u)$, deploys a monitor in the node u. More formally,

$$Add(S, u) \leftarrow S \cup \{u\}.$$

The first stage of the shake procedure randomly selects k nodes where a monitor is deployed and performs the *Drop* move over them. Then, in the second stage, monitors are deployed using the *Add* move until all the links are covered. We propose two different strategies to perform the second stage, resulting in two shake procedures.

On the one hand, the first strategy, named *RandomShake*, follows the traditional VNS scheme by randomly selecting k nodes that are not in S and deploying a monitor in them with the *Add* movement. Notice that this strategy usually produces solutions in which there are more monitors than necessary since the candidate nodes to host a monitor are selected completely at random.

On the other hand, we propose a new shake method, named *IntensifiedShake*, that performs this second stage intelligently, choosing the most promising nodes to host a monitor. It is necessary to establish a criterion to decide which the next node selected to host a monitor is. In this work, we propose to use the same criterion as the one used in *GreedyConstructive*, that is, the selected nodes are those whose links that are not covered generate the largest penalty (always excluding the leaf nodes). This second strategy has not been widely explored in the literature, but it is an interesting approach since it is able to find promising regions of the search space without compromising the diversity of the search, thus leading to more promising solutions. Therefore, it is a strategy to be considered in those optimization problems where it is possible to consider a greedy criterion in the shake procedure.

4 | COMPUTATIONAL RESULTS

This section has two main goals. First of all, it is necessary to adjust the input parameters of the proposed algorithm, as well as the algorithmic strategies that conform the final method. Having configured the best variant of the proposed procedure, the second objective is to perform a competitive testing considering the state-of-the-art method for the MPP.

All the algorithms have been implemented in Java 11 and the experiments have been conducted in an AMD EPYC 7282 (2.8 GHz) and 72 GB RAM. We have considered the same set of instances as the ones presented in the related literature, where the best algorithm for the MPP is introduced [20]. From the complete set of instances, we have not been able to consider `delaunay_n19` and `delaunay_n20` due to hardware constraints, since our computer does not have enough memory to store them (although the algorithm would be able to solve them). However, the results obtained with the largest instances will show how the improvements obtained by the proposed algorithm increase with the size of the instance. The final set of instance consists of 37 instances. In order to favor future comparisons, we have made publicly available these instances at https://grafo.etsii.urjc.es/MonitorPlacement.

All the experiments report the following metrics: Avg., the average objective function value obtained by each algorithm among the considered instances; time (s), the average computing time required by each algorithm to finish, measured in seconds; Dev (%), the average deviation of the objective function value with respect to the best solution found during the experiment, evaluated as $\frac{|Best-Obj.F.|}{Best}$ for each instance; and # Best, the number of times that the algorithm reaches the best solution of the algorithm in the experiment. In all the experiments, the best result for each metric is highlighted in bold font.

In order to select the best values for each parameter, some preliminary experiments have been executed. In all of them (from Tables 1–5), 10 representative instances have been used to configure the proposed algorithm in order to avoid overfitting. Those instances are: `delaunay_n10`, `delaunay_n11`, `frb30-15-3`, `frb35-17-1`, `frb40-19-2`, `p2p-Gnutella04`, `p2p-Gnutella05`, `p2p-Gnutella06`, `tech-routers-rf`, `tech-WHOIS`.

The first preliminary experiment is intended to evaluate the performance of the proposed constructive methods, *Greedy-Constructive* and *GreedyDestructive*. Since they are greedy algorithms, we have constructed a single solution for each instance using each constructive procedure, as presented in Table 1.

As suggested in Section 3.2, the destructive procedure is considerably faster than the constructive one, since the final solutions contain a monitor deployed in most of the nodes. Therefore, the number of nodes to be removed when considering a monitor in each node (*GreedyDestructive*) is rather smaller than the number of nodes to be added when considering an empty initial solution (*GreedyConstructive*). However, in terms of quality, the solutions generated with *GreedyConstructive* are slightly better than the ones created with *GreedyDestructive*, as indicated by the small deviation presented by the *GreedyDestructive* procedure. Analyzing these results, it is not possible to determine which is the best constructive procedure, so in a latter experiment the performance of the constructive procedures coupled with the local search method will be tested.

TABLE 1 Results obtained when creating a single solution for each instance by each constructive procedure

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GreedyDestructive	1844.90	**0.19**	0.52	2
GreedyConstructive	**1837.80**	1.82	**0.04**	**8**

TABLE 2 Heat map of the computing times (left) and average deviation (right) when considering different values of δ and k_{max} for *RandomShake*

	k_{max}				
δ	0.1	0.2	0.3	0.4	0.5
1	0.63	0.93	1.86	2.41	3.92
10	2.17	6.02	11.28	15.02	27.95
20	4.04	12.12	21.56	29.29	54.20
30	6.04	17.84	31.70	42.69	80.46
40	7.67	22.96	85.93	82.07	107.57

	k_{max}				
δ	0.1	0.2	0.3	0.4	0.5
1	0.23	0.23	0.22	0.22	0.22
10	0.10	0.15	0.07	0.10	0.05
20	0.02	0.10	0.05	0.09	0.05
30	0.02	0.10	0.05	0.04	0.05
40	0.02	0.09	0.05	0.02	0.05

TABLE 3 Heat map of the computing times (left) and average deviation (right) when considering different values of δ and $k_{\max}$ for *IntensifiedShake*

	$k_{\max}$				
δ	0.1	0.2	0.3	0.4	0.5
1	0.20	0.26	0.57	1.15	2.04
10	0.19	0.99	3.15	6.05	12.74
20	0.20	1.89	5.75	11.79	22.25
30	0.23	2.56	8.50	17.91	33.51
40	0.24	3.22	10.93	27.10	45.64

	$k_{\max}$				
δ	0.1	0.2	0.3	0.4	0.5
1	0.36	0.34	0.32	0.12	0.12
10	0.36	0.17	0.07	0.11	0.06
20	0.36	0.09	0.03	0.11	0.06
30	0.33	0.09	0.03	0.10	0.06
40	0.33	0.09	0.03	0.10	0.03

TABLE 4 Comparison of both shake strategies for the BVNS algorithm with the corresponding best values for each search parameter

Shake method	Avg.	Time (s)	Dev. (%)	#Best
IntensifiedShake	**1816.20**	**1.89**	**0.01**	**9**
RandomShake	1818.80	4.04	0.22	6

TABLE 5 Comparison of the contribution of each component of the algorithm proposed

Algorithm	Avg.	Time (s)	Dev. (%)	#Best
GreedyDestructive	1844.90	**0.19**	1.89	0
GreedyDestructive+ILS	1819.30	0.21	0.26	4
BVNS	**1816.20**	1.89	**0.00**	**10**

The second experiment evaluates the influence of considering the intelligent local search ILS instead of executing the exhaustive local search ELS, both described in Section 3.3. Figure 3 shows the comparison of the computing times for executing the ILS and ELS over the solution generated by the *GreedyConstructive* procedure to decide if it is worth considering ILS over ELS.

Notice that the figure is represented using a logarithmic scale due to the vast differences between the computing times of both local search methods. It is worth mentioning that both local search methods result in solutions of similar quality, so we only analyze computing times. Regarding these results, we can clearly see the positive impact of considering the intelligent local search. On average, it is 2500 times faster than ELS, reaching a speedup of 5200 in the most complex instances. In particular, notice that ILS does not require more than 10 s in any of the considered instances. Furthermore, in the most complex instances, namely p2p-Gnutella04, p2p-Gnutella05, and p2p-Gnutella06, the ELS has been stopped when reaching 100 000 s, highlighting the relevance of considering the intelligent local search procedure. Therefore, we consider ILS as the local search method for the remaining experiments.

The next experiment tries to settle which the most adequate constructive procedure for the MPP is. As stated above, it is not clear which the best method to generate the initial solution is, so we test the performance of both greedy procedures executed isolated and when they are coupled with ILS. Figure 4 shows the relation between computing time and average deviation when considering these four procedures.

The first conclusion that can be extracted from this experiment is that ILS considerably improves the generated solution without requiring a high computational effort in both greedy procedures. Additionally, we can see how *GreedyDestructive+ILS* is able to outperform the results provided by *GreedyConstructive+ILS*, requiring much less computing time. Therefore, *Greedy-Destructive* procedure is used as the method for generating the solution used as starting point for the complete BVNS algorithm. These results indicates that it is interesting to consider a destructive approach which starts with all elements selected in those optimization problems in which high-quality solutions have most of the elements selected, since it is able to considerably reduce computing time without deteriorating the quality of the solution. Additionally, the effect of ILS with respect to ELS suggests that it is recommended to include information about the structure of the solution in the local search if it is available.

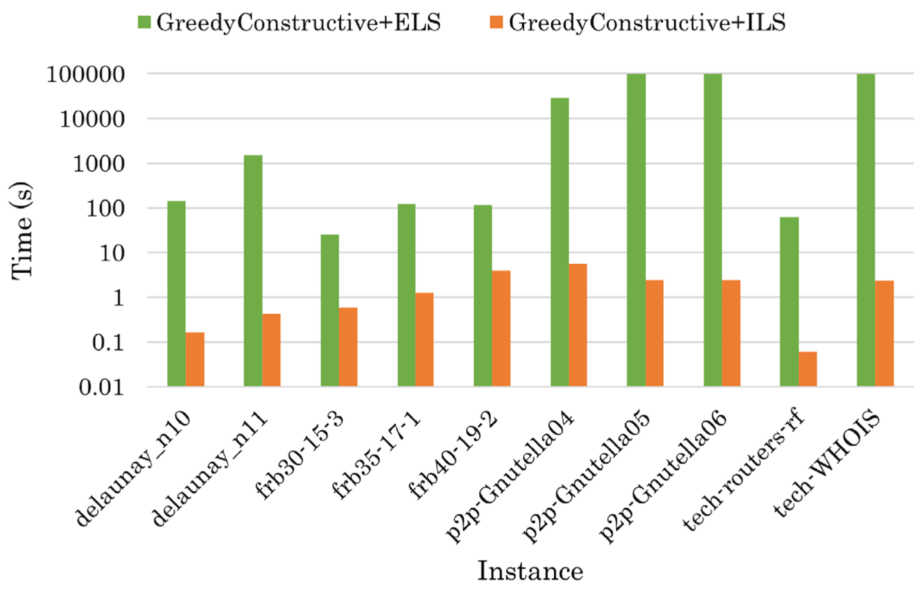

FIGURE 3 Comparison of the time required by the local search and the efficient variant proposed for each instance in the preliminary set

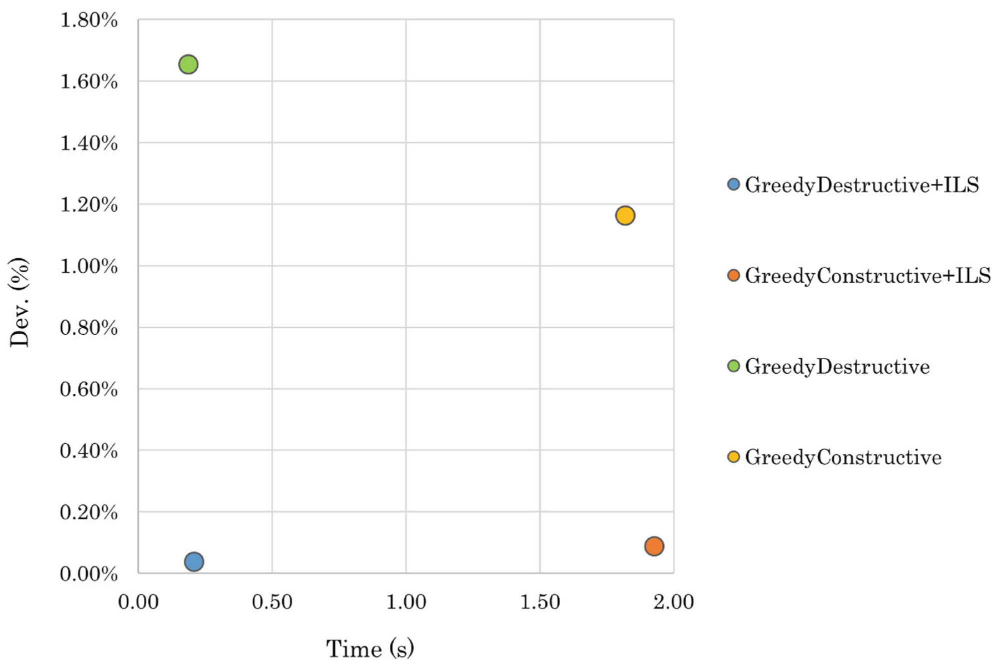

FIGURE 4 Comparison of the time and the average deviation between both construction strategies isolated and then considering ILS

Having defined the constructive procedure and the local search method, we need to tune the values for the input parameters of BVNS. The algorithm requires just two parameters: δ, which is the number of complete VNS iterations, and $k_{\max}$, the maximum neighborhood to be explored in each complete iteration. The values tested for the largest neighborhood are $k_{\max} = \{0.1, 0.2, 0.3, 0.4, 0.5\}$, where each value represents a percentage of the number of nodes of the instance to guarantee the scalability of the proposal. We do not consider larger values of $k_{\max}$ since perturbing more than half of the solution will result in a completely different one, which is against the philosophy of the VNS framework. Regarding the number of iterations, we test $\delta = \{1, 10, 20, 30, 40\}$.

This work proposes two different shake methods, *RandomShake* and *IntensifiedShake*, and the best values for these parameters for one shake method are not necessarily the best values for the other shake method. Therefore, we have conducted two equivalent experiments varying the shake method considered. Since modifying one of the parameters might affect to the other one, we have decided to show the results obtained in two heat maps where the worst values are colored in red and the best values in green, interpolating the values between them using a color gradient. Table 2 shows the results obtained by considering *RandomShake*.

Analyzing the heat map of computing times (left), as expected, it grows with the number of iterations and maximum neighborhood explored, being two orders of magnitude slower than the fastest option ($k_{\max} = 0.1$ and $\delta = 1$) in some cases. If we simultaneously analyze the heat map of average deviation (right), we can clearly see that the best values are obtained when considering small values of $k_{\max}$, specifically $k_{\max} = 0.1$, and the number of iterations stagnates when reaching 20. Since the computing time of $k_{\max} = 0.1$ and $\delta = 20$ is also one of the smallest in the experiment, we select these parameter values when considering *RandomShake*. Table 3 now shows the same experiment but considering *IntensifiedShake*.

Analogously to the previous experiment, the computing time also gets increased with the value of $k_{\max}$ and δ, although the differences are not so remarkable. Regarding the average deviation, the best values are obtained with $k_{\max} = 0.3$, which indicates that the intensified shake is able to increase the size of the maximum neighborhood without deteriorating the quality of the generated solutions. With respect to the number of iterations, it seems that performing more than 20 iterations does not result in any improvement, so we select $k_{\max} = 0.2$ and $\delta = 20$ for *IntensifiedShake*. The rationale behind this is that, although considering the next $k_{\max}$ value reduces the deviation from 0.09 to 0.03, we do not believe that it is a significant improvement for multiplying the computing time by 3.

Having defined the best parameter values for $k_{\max}$ and δ for both *RandomShake* and *IntensifiedShake*, it is necessary to compare the performance of each shake method in the BVNS algorithm. Table 4 shows the results obtained by both variants.

As it can be seen in the results, *IntensifiedShake* requires less computing time than *RandomShake*. This behavior can be partially explained since the solution to which the local search is applied is closer to a local optimum in the case of *IntensifiedShake* than in *RandomShake*, reducing the number of iterations required to reach the local optimum. Additionally, *IntensifiedShake* is able to outperform *RandomShake* in number of best solutions and in average deviation. Notice that *IntensifiedShake* misses just one best value, remaining really close to it as derived from a deviation of 0.01%. It is worth mentioning that *RandomShake* is rather competitive in this case, but we select *IntensifiedShake* for being slightly better in all the metrics requiring considerably less computing time than *RandomShake*. In this case, the greedy selection of elements in *IntensifiedShake* is able to reduce computing time without deteriorating the objective function value, which highlights the relevance of considering this strategy for optimization problems in which it is possible to define a greedy criterion to select an element to be included in the solution.

The last preliminary experiment is designed to evaluate the contribution of each component of the final algorithm to the quality of the generated solutions. To that end, we compare the constructive method isolated, then coupled with the intelligent local search and, finally, the complete *BVNS* algorithm, configured with *GreedyDestructive*, *ILS*, *IntensifiedShake*, $k_{\max} = 0.2$, and $\delta = 20$. Table 5 shows the results obtained in this experiment.

It is worth mentioning that the efficient design of each component leads *BVNS* to require comparable computing time than the constructive procedure isolated or coupled with the local search method. Although ILS allows the algorithm to find 4 best solutions, it is the complete *BVNS* algorithm the one that is able to reach all the best solutions. However, the small deviation achieved by the combination of constructive and local search methods result in a simple yet effective algorithm to provide high quality solutions in negligible computing times. These results show the contribution of each part of the proposed algorithm to the final version.

At this point, all the components and parameters of the *BVNS* algorithm have been fixed, so the next experiment is devoted to evaluate the performance of *BVNS* when comparing it with a hybrid search evolutionary algorithm (LS+PI) EA, one of the most widely used algorithms found in the state of the art [20], which consists of an effective hybrid search heuristic that leverages the combination of the greedy local search method with evolution-based heuristics. The complete set of 37 instances is considered in this final experiment, grouping them by type. Table 6 shows the results obtained when considering the complete testbed of instances. Notice that the column Median O.F. reports the median objective function value obtained along 100 executions for each instance, considering the same experimental methodology described in [20]. It is worth mentioning that the hardware used to perform the experiments in this research is directly comparable with the one considered in the state of the art. In particular, the score given in a CPU benchmark to each processor is the same (1.9).[1]

Results in Table 6 show how *BVNS* is able to find 37 best solutions, that is, in each type of instances the algorithm reaches the best solution for every instance. If we now look at the time required by each algorithm, in the type of instances where the difference is the largest (frb), *BVNS* is 445 times faster than *(LS+PI) EA*. Looking at the average deviation obtained by each algorithm in each type of instances, it can be seen that *BVNS* value is always 0.0%, while the state-of-the-art algorithm has an average deviation of approximately 17.0% taking into account all types of instances, reaching the maximum value in `delaunay` (40.47%) and `internet-as` (33.22%) instances, and the minimum value in `frb` instances (0.66%). Notice that, in this last type of instances, the difference in time between the two algorithms is the greatest one. It is worth mentioning that the largest differences in deviation are obtained in the most complex instances, highlighting the scalability of the proposed algorithm. Analyzing these results, we can conclude that the proposed BVNS is a competitive algorithm for solving the MPP, requiring a reduced computational effort even for the most complex instances, we refer the reader to Table A1 to view individual results.

[1]https://www.cpubenchmark.net/compare/Intel-Xeon-E5-2690-v4-vs-AMD-EPYC-7282/2780vs3625

TABLE 6 Comparison of the hybrid search evolutionary algorithm (LS+PI) EA and the proposed BVNS for each instance type

Instance type	Algorithm	Median O.F.	Avg. time (s)	Dev. (%)	#Best
delaunay	***BVNS***	**41 311.72**	**272.21**	**0.00**	**9**
	(LS+PI) EA	100 602.33	475.78	40.47	0
frb	***BVNS***	**584.67**	**0.56**	**0.00**	**15**
	(LS+PI) EA	588.67	252.56	0.66	0
internet-as	***BVNS***	**5701.00**	**19.95**	**0.00**	**1**
	(LS+PI) EA	7595.00	179.18	33.22	0
NREN	***BVNS***	**422.00**	**0.05**	**0.00**	**1**
	(LS+PI) EA	452.00	2.02	7.11	0
p2p-Gnutella	***BVNS***	**6001.67**	**6.49**	**0.00**	**9**
	(LS+PI) EA	7966.67	71.95	24.58	0
tech	***BVNS***	**1540.50**	**1.98**	**0.00**	**2**
	(LS+PI) EA	1658.50	32.08	7.33	0

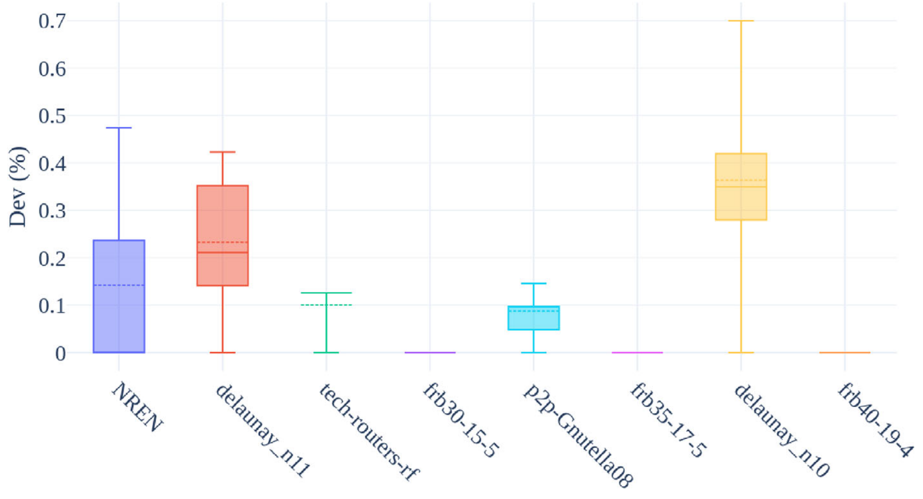

FIGURE 5 Box-plots for some representative instances to show the variability of the proposed algorithm

In order to evaluate if there are statistically significant differences between the results obtained by both algorithms, the pairwise nonparametric Wilcoxon Signed Ranks statistical test is performed. The resulting p-value smaller than 0.001 supports the hypothesis that both samples belong to different populations.

Finally, to evaluate the effect of the randomness in the proposed algorithm, Figure 5 shows the box-plot obtained for the 100 executions of BVNS. In order to present all the box-plots in the same figure, the reported values are the deviation with respect to the best value of the 100 executions per instance. The considered instances are the same as the ones presented in the best previous work [20] to facilitate the comparison. As it can be seen in the figure, most of them present minimum or even zero variability, supporting the robustness of the proposal. The largest variability is obtained in the most complex instances, where the objective function value varies in a range of five units, which is negligible when considering the magnitude of the objective function value. We refer the reader to Table A2 in the Appendix for the individual results on every instance.

5 | CONCLUSIONS

This work deals with the MPP, which tries to locate monitors in a network in such a way that all the communications are surveyed to protect the network from external attacks or failures. Since it is a critical task, it requires a fast response, so the proposed algorithm must be able to perform the monitoring without requiring large computing times. The problem is addressed by considering the BVNS framework, proposing an intelligent local search that is able to leverage the identification of promising regions of the search space to minimize the impact in the computing time without deteriorating the quality of the solutions provided.

Furthermore, two greedy procedures are proposed to produce a promising initial solution. Both methods follow opposite directions: on the one hand, the *GreedyConstructive* starts from scratch and constructs a solution by locating monitors; on the other hand, a novel approach is followed by *GreedyDestructive*, which initially considers that a monitor is deployed in every node and iteratively removes monitors while maintaining the network covered. Finally, an intensified shake is proposed, which is able to perform an intelligent selection of the nodes inside the shake procedure of VNS with the aim of guiding the search to a more promising region of the search space. The experimental results show how every component of the proposed algorithm has a positive effect in the final results, emerging BVNS as a competitive method for solving the MPP.

Future lines of research comprehend the study of the strategies presented in this work when applying it to problems related to the monitor placement but with additional constraints, such as weighted nodes, fault tolerance, and so forth, to evaluate the adaptation of the proposed algorithm.

ACKNOWLEDGMENTS

Alejandra Casado, Jesús Sánchez-Oro, and Abraham Duarte research was funded by "Ministerio de Ciencia, Innovación y Universidades" under grant ref. PGC2018-095322-B-C22, "Comunidad de Madrid" and "Fondos Estructurales" of European Union with grant refs. S2018/TCS-4566 and Y2018/EMT-5062. Nenad Mladenović has been partially supported by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan, Grant No. AP08856034.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in MonitorPlacement at https://grafo.etsii.urjc.es/MonitorPlacement.

ORCID

Alejandra Casado https://orcid.org/0000-0003-3417-6859
Nenad Mladenović https://orcid.org/0000-0001-6655-0409
Jesús Sánchez-Oro https://orcid.org/0000-0003-1702-4941
Abraham Duarte https://orcid.org/0000-0002-4532-3124

REFERENCES

[1] G. Andersson, P. Donalek, R. Farmer, N. Hatziargyriou, I. Kamwa, P. Kundur, N. Martins, J. Paserba, P. Pourbeik, and J. Sanchez-Gasca, *Causes of the 2003 major grid blackouts in north america and europe, and recommended means to improve system dynamic performance*, IEEE Trans. Power Syst. **20** (2005), no. 4, 1922–1928.
[2] T. Back and S. Khuri, *An evolutionary heuristic for the maximum independent set problem*, Proc. 1st IEEE Conf. Evol. Comput. IEEE World Congr. Comput. Intell., IEEE, 1994, pp. 531–535.
[3] S. P. Borgatti and M. G. Everett, *A graph-theoretic perspective on centrality*, Soc. Netw. **28** (2006), no. 4, 466–484.
[4] P. Casas-Martínez, A. Casado-Ceballos, J. Sánchez-Oro, and E. G. Pardo, *Multi-objective grasp for maximizing diversity*, Electronics **10** (2021), no. 11, 1232.
[5] D. P. Chandu, *A parallel genetic algorithm for three dimensional bin packing with heterogeneous bins*. arXiv preprint arXiv:1411.4565, 2014.
[6] P. Crucitti, V. Latora, and M. Marchiori, *Model for cascading failures in complex networks*, Phys. Rev. E **69** (2004), no. 4, 045104.
[7] I. Dinur and S. Safra, *On the hardness of approximating minimum vertex cover*, Ann. Math. **162** (2005), 439–485.
[8] A. Duarte, R. Martí, F. Glover, and F. Gortazar, *Hybrid scatter tabu search for unconstrained global optimization*, Ann. Oper. Res. **183** (2011), no. 1, 95–123.
[9] I. K Evans, Evolutionary algorithms for vertex cover. Proc. Int. Conf. Evol. Program., Springer, 1998, pp. 377–386.
[10] P. Hansen, N. Mladenović, R. Todosijević, and Saı¨ d Hanafi, *Variable neighborhood search: Basics and variants*, EURO J. Comput. Optim. **5** (2017), no. 3, 423–454.
[11] N. Hatano and M. Suzuki, *Quantum annealing and other optimization methods*. arXiv preprint math-ph/0506007, 2005.
[12] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, MIT Press, Massachussetts, 1992.
[13] R. M. Karp, "*Reducibility among combinatorial problems*," *Complexity of computer computations*, Springer, New York, 1972, pp. 85–103.
[14] S. Lagraa and J. François, *Knowledge discovery of port scans from darknet*, Proc. 2017 IFIP/IEEE Symp. Integr. Netw. Serv Manag. (IM). IEEE, 2017, pp. 935–940.
[15] E. L. Lawler and D. E. Wood, *Branch-and-bound methods: A survey*, Oper. Res. **14** (1966), no. 4, 699–719.
[16] R Luling and B. Monien, *Load balancing for distributed branch & bound algorithms*, Proc. 6th Int. Parallel Process. Symp., IEEE, 1992, pp. 543–548.
[17] M. Milanovic, *Solving the generalized vertex cover problem by genetic algorithm*, Comput. Inform. **29** (2010), no. 6, 1251–1265.
[18] R. Mueller-Bady, R. Gad, M. Kappes, and I. Medina-Bulo, *Using genetic algorithms for deadline-constrained monitor selection in dynamic computer networks*, Proc. Comp. Publ. 2015 Annu. Conf. Genet. Evol. Comput, 2015, pp. 867–874.
[19] R. Mueller-Bady, M. Kappes, I. Medina-Bulo, and F. Palomo-Lozano, *Optimization of monitoring in dynamic communication networks using a hybrid evolutionary algorithm*, Proc. Genet. Evol. Comput. Conf., 2017, pp. 1200–1207.
[20] R. Mueller-Bady, M. Kappes, I. Medina-Bulo, and F. Palomo-Lozano, *An evolutionary hybrid search heuristic for monitor placement in communication networks*, J. Heurist. **25** (2019), no. 6, 861–899.
[21] M. E. J. Newman, *A measure of betweenness centrality based on random walks*, Soc. Netw. **27** (2005), no. 1, 39–54.
[22] M. Pelikan and D. E. Goldberg, "*Hierarchical bayesian optimization algorithm*," *Scalable optimization via probabilistic modeling*, Springer, New York, 2006, pp. 63–90.
[23] S. Pérez-Peló, J. Sánchez-Oro, A. Gonzalez-Pardo, and A. Duarte, *A fast variable neighborhood search approach for multi-objective community detection*, Appl. Soft Comput. **112** (2021), 107838.

[24] J. Sánchez-Oro, J. J. Pantrigo, and A. Duarte, *Combining intensification and diversification strategies in vns. an application to the vertex separation problem*, Comput. Oper. Res. **52** (2014), 209–219.
[25] D. Zhang, E. K. Cetinkaya, and J. P. G. Sterbenz, *Robustness of mobile ad hoc networks under centrality-based attacks*, Proc. 2013 5th Int. Congr. Ultra Modern Telecommun. Control Syst. Worksh. (ICUMT)., IEEE, 2013, pp. 229–235.

How to cite this article: A. Casado, N. Mladenović, J. Sánchez-Oro, and A. Duarte, *Variable neighborhood search approach with intensified shake for monitor placement*, Networks.. (2022), 1–15. https://doi.org/10.1002/net.22134

APPENDIX

TABLE A1 Comparison of (LS+PI) EA [20] and BVNS for each instance, considering the median objective function value obtained along 100 executions, as well as the average computing time

	BVNS		(LS+PI) EA	
Instance	Time (s)	Median O.F.	Time (s)	Median O.F.
delaunay_n10	0.20	717.50	3.33	755
delaunay_n11	0.42	1422.00	6.79	1514
delaunay_n12	2.16	2863.00	13.46	3046
delaunay_n13	9.63	5700.50	25.11	6120
delaunay_n14	37.75	11 415.50	54.10	12 304
delaunay_n15	174.57	22 821.50	178.13	24 848
delaunay_n16	484.26	45 687.50	472.69	51 058
delaunay_n17	262.56	91 917.00	1106.82	151 210
delaunay_n18	1478.32	189 261.00	2421.55	654 566
frb30-15-1	0.22	435.00	63.30	438
frb30-15-2	0.21	435.00	72.85	437
frb30-15-3	0.25	435.00	67.55	438
frb30-15-4	0.20	435.00	72.84	437
frb30-15-5	0.18	435.00	77.69	436
frb30-17-1	0.55	578.00	230.92	582
frb30-17-2	0.43	578.00	259.31	583
frb30-17-3	0.46	578.00	218.72	582
frb30-17-4	0.47	578.00	236.29	582
frb30-17-5	0.58	578.00	273.25	582
frb40-19-1	0.76	741.00	413.36	747
frb40-19-2	0.99	741.00	414.94	747
frb40-19-3	0.90	741.00	484.33	747
frb40-19-4	1.24	741.00	473.27	745
frb40-19-5	0.93	741.00	429.84	747
internet-as	19.94	5701.00	179.18	7595
nren	26.89	422.00	2.02	452
p2p-Gnutella04	0.02	4352.50	24.69	5017
p2p-Gnutella05	5.58	3431.00	21.46	3958
p2p-Gnutella06	3.36	3407.00	22.88	3916
p2p-Gnutella08	3.44	2057.00	14.10	2344
p2p-Gnutella09	1.02	2574.00	17.19	2984
p2p-Gnutella24	1.52	7210.00	67.03	9382
p2p-Gnutella25	11.76	6017.00	36.37	7794
p2p-Gnutella30	7.15	9272.50	123.85	12 419
p2p-Gnutella31	24.52	15 694.00	320.01	23 886
tech-routers-rf	0.14	796.00	7.05	849
tech-WHOIS	3.82	2285.00	57.10	2468

TABLE A2 Individual results found by algorithm BVNS for each considered instance, considering best, worst, and average objective function value, as well as the average computing time.

Instance	Min. O.F.	Max. O.F.	Avg. O.F.	Avg. time (s)
delaunay_n10	715	720	717.60	0.20
delaunay_n11	1419	1425	1422.30	0.42
delaunay_n12	2853	2868	2861.60	2.16
delaunay_n13	5698	5708	5701.70	9.64
delaunay_n14	11 408	11 426	11 416.40	37.75
delaunay_n15	22 809	22 833	22 821.30	174.58
delaunay_n16	45 662	45 714	45 688.90	484.26
delaunay_n17	91 915	92 023	91 938.80	262.57
delaunay_n18	189 261	189 261	189 261.00	1478.33
frb30-15-1	435	435	435.00	0.23
frb30-15-2	435	435	435.00	0.22
frb30-15-3	435	435	435.00	0.25
frb30-15-4	435	435	435.00	0.20
frb30-15-5	435	435	435.00	0.18
frb35-17-1	578	578	578.00	0.56
frb35-17-2	578	578	578.00	0.43
frb35-17-3	578	578	578.00	0.46
frb35-17-4	578	578	578.00	0.48
frb35-17-5	578	578	578.00	0.59
frb40-19-1	741	741	741.00	0.76
frb40-19-2	741	741	741.00	1.00
frb40-19-3	741	741	741.00	0.91
frb40-19-4	741	741	741.00	1.24
frb40-19-5	741	741	741.00	0.94
internet-as	5701	5704	5701.70	19.95
nren	422	424	422.60	26.89
p2p-Gnutella04	4351	4355	4352.80	0.02
p2p-Gnutella05	3430	3432	3430.90	5.58
p2p-Gnutella06	3406	3410	3407.20	3.36
p2p-Gnutella08	2055	2058	2056.80	3.45
p2p-Gnutella09	2574	2574	2574.00	1.02
p2p-Gnutella24	7209	7212	7210.20	1.52
p2p-Gnutella25	6017	6017	6017.00	11.76
p2p-Gnutella30	9271	9274	9272.30	7.16
p2p-Gnutella31	15 694	15 695	15 694.30	24.52
tech-routers-rf	795	796	795.80	0.14
tech-WHOIS	2285	2286	2285.30	3.83