




WILEY

INTERNATIONAL
TRANSACTIONS
IN OPERATIONAL
RESEARCHIntl. Trans. in Op. Res. 0 (2024) 1–21
DOI: 10.1111/itor.13468

A variable neighborhood search for the median location problem with interconnected facilities

I. Lozano-Osorio^a , J. Sánchez-Oro^{a,*} , A.D. López-Sánchez^b 
and A. Duarte^a ^aUniversidad Rey Juan Carlos/ Tulipán s/n., 28933, Móstoles, Madrid, Spain^bUniversidad Pablo de Olavide/ Ctra., de Utrera km 1, Sevilla 41013, SpainE-mail: isaac.lozano@urjc.es [Lozano-Osorio]; jesus.sanchezoro@urjc.es [Sánchez-Oro];
adlopsan@upo.es [López-Sánchez]; abraham.duarte@urjc.es [Duarte]

Received 20 December 2022; received in revised form 8 February 2024; accepted 13 April 2024

Abstract

The p -median problem has been widely studied in the literature. However, there are several variants that include new constraints to the classical problem that make it more realistic. In this work, we study the variant that considers interconnected facilities, that is, the distances between each pair of facilities are less than or equal to a certain threshold r . This optimization problem, usually known as the median location problem with interconnected facilities, consists of locating a set of interconnected facilities to minimize the distance between those interconnected facilities and the demand points. The large variety of real-world applications that fit into this model makes it attractive to design an algorithm able to solve the problem efficiently. To this end, a procedure based on the variable neighborhood search methodology is designed and implemented by using problem-dependent neighborhoods. Experimental results show that our proposal is able to reach most of the optimal solutions when they are known. Additionally, it outperforms previous state-of-the-art methods in those instances where the optima are unknown. These results are further confirmed by conducting nonparametrical statistical tests.

Keywords: facility location; p -median problem; interconnected facilities; variable neighborhood search

1. Introduction

In general terms, facility location problems (FLPs) identify a family of optimization problems that consist of finding the best location to host a set of facilities (among a set of potential facility locations), subject to the constraint that every demand point must be serviced by the nearest hosted facility. Therefore, the objective is to minimize the total cost. In the literature, there exist many variants of facility location problems given the importance of a correct emplacement of them,

© 2024 The Authors.

International Transactions in Operational Research published by John Wiley & Sons Ltd on behalf of International Federation of Operational Research Societies.

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.

depending on the criterion to be optimized and the constraints to be considered. In addition, practitioners are forced to include new constraints in order to model more realistic situations.

A well-known facility location problem is the p -median problem, first studied by Hakimi (1964, 1965), that seeks to minimize the total (or average) cost (or distance) between demand points and their nearest facility. We refer the reader to Reese (2006) where the history of this problem is described, and, furthermore, an overview of the techniques that have been applied to solve the p -median problem is included. Some of the variants most studied are the capacitated p -median problem (Tuba et al., 2018) or the obnoxious p -median problem (Lin et al., 2018) and (Herrán et al., 2018), (Mousavi et al., 2023), among others.

This paper addresses a variant of the p -median problem which considers that the selected facilities must be interconnected, satisfying a distance threshold between each pair of them. This problem is named the median location problem with interconnected facilities, from now on MPIF, which includes a constraint that indicates that the distance between two facilities cannot exceed a given threshold r . MPIF was originally introduced by Cherklesly et al. (2019), where many realistic problems that fit this model were described. MPIF has been proven to be \mathcal{NP} -hard in Cherklesly et al. (2019). For instance, in the context of the Internet of Things, the deployment of a network of heterogeneous sensors, such as alarms or motion detectors, among others, in a given area must be done in such a way that communication through the network is guaranteed without loss of quality (Srinidhi et al., 2019). Another example appears when it is necessary to locate forest fire stations as described in Cherklesly et al. (2019) in order to minimize the impact of a natural disaster.

MPIF has been approached by considering both, exact and heuristic approaches. In particular, Cherklesly et al. (2019) proposed three exact formulations for the MPIF, being able to optimally solve small- and medium-sized instances (with $p \leq 500$) in less than a minute (on average). Although the authors proposed three different formulations, they stated that the one named MPIF-2 is the state of the art for MPIF. We refer the reader to the original work for a complete and detailed description of this mathematical model. Cherklesly et al. (2019) also proposed a hybrid metaheuristic based on the iterated local search (ILS) methodology (see Lourenço et al., 2003) to deal with large instances (with $p \leq 900$). Specifically, the algorithm builds an initial feasible solution with a constructive method inspired by the greedy randomized adaptive search procedure (GRASP) by Feo and Resende (1995), which is further improved by typical ILS strategies (perturbing a solution and then improving it with a local search method). The perturbation strategy removes one located facility and incorporates a nonlocated facility that fulfills the distance criterion. Notice that this process is repeated η times, with $\eta < p$, in order to have a perturbation of magnitude η . The improvement phase uses three different neighborhood operators: an exchange operator that selects one located facility from the solution and another nonlocated facility that is not in the solution and interchanges both; a removal operator that selects the facility yielding the largest decrease in the cost obtaining an unfeasible solution; and an insertion operator that aims to increase the number of facilities by inserting one facility not currently selected with the largest decrease in terms of cost.

In this paper, we propose an effective and efficient variable neighborhood search (VNS) algorithm, first proposed by Mladenović and Hansen (1997), to solve the MPIF. This methodology has been successfully applied to a wide variety of hard combinatorial optimization problems Hansen et al. (2017). It is worth mentioning that VNS was already used to successfully solve the p -median problem more than 20 years ago by Hansen and Mladenović (1997). Additionally, García-López

et al. (2002) solved the classical p -median problem by considering parallel strategies within the VNS methodology.

The main contributions of this work are the following:

- A greedy constructive procedure based on the contribution of each element to the objective function to generate high-quality initial solutions.
- The computational complexity of the constructive procedure is reduced by storing specific information about the solution in a memory structure.
- An efficient and effective VNS algorithm for solving the MPIF is designed.
- The shake procedure is focused on exploring only those solutions that are feasible after a single move, thus reducing the computational effort without affecting the quality of the results.
- The effect of each component of the proposed algorithm is carefully analyzed to verify the contribution of each stage to the final results.
- The dataset is increased with more complex and challenging instances, where the optimal values are unknown, and exact algorithms are unable to find a solution.
- The complete source code of the proposed algorithm is publicly available to ease further comparisons as well as the complete results.¹

The rest of the paper is organized as follows: Section 2 defines the facility location problem addressed in this paper. Section 3 describes the proposed algorithm and the new strategies that we have implemented to solve it. Section 4 includes the computational results. Finally, the conclusions and future research are discussed in Section 5.

2. Problem definition

Let $G = (N, E)$ be a weighted graph where N identifies the set of nodes and E represents the set of edges. In FLPs the set of nodes is usually composed of two different subsets $N = (F, D)$, where F is the set of candidate facilities and D is the set of demand points. Each edge is defined as $(i, j) \in E$, with $i, j \in N$, where the weight is represented by w_{ij} . Figure 1 illustrates a graph example with eight nodes with $F = \{0, A, C, D, F\}$ and $D = \{B, E, G\}$; and nine edges with the weight between each pair of nodes represented with a number close to each edge.

In the classical p -median problem, a feasible solution consists of selecting p facilities in F , where each demand point is mapped to a single facility, and the sum of the weights between all demand points and the corresponding facilities is minimized. The MPIF (see Cherklesly et al., 2019) incorporates an additional constraint to force that the set of selected facilities are interconnected. More formally, given a solution S with p facilities, the sum of weights between each facility $j \in S$ and, at least, one of the remaining $p - 1$ facilities in $S \setminus \{j\}$, is smaller than or equal to a threshold value r . In mathematical terms,

$$\min_{j' \in S \setminus \{j\}} d(j, j') \leq r \quad \forall j \in S. \quad (1)$$

¹<https://grafo.etsii.urjc.es/MPIF>

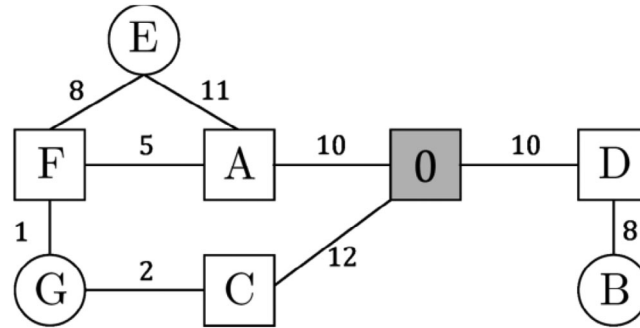


Fig. 1. Example with eight nodes and nine edges where the set of candidate facilities are represented with a square shape while the demand points are depicted as a circle.

Naturally, $d(j, j')$ is the shortest path between j and j' . Notice that, due to the constraint of interconnection among facilities, any solution for the MPIF is always conformed by a single connected component.

As stated in Cherkesly et al. (2019), the real application of the MPIF considers that there must always be a root node. For the sake of simplicity, it is usually represented by 0 and always belongs to any feasible solution. The root node indicates that the selected facilities should be interconnected with such a node to ensure that, in turn, will be interconnected with other selected facilities.

The objective of the MPIF is then to locate a set of interconnected facilities, $S \subset F$, with $|S| = p$, where p is a constraint of the problem determined a priori, to minimize the total sum of weights between each demand point $i \in D$ and its closest selected facility $j \in S$. Therefore, given a solution S , the objective function of MPIF is evaluated as follows:

$$MPIF(S) \leftarrow \sum_{i \in D} \min_{j \in S} d(i, j), \quad (2)$$

where $d(i, j)$ is the shortest path between i and j .

The optimization problem then consists of finding a solution S^* with the minimum MPIF value. More formally,

$$S^* \leftarrow \arg \min_{S \in \mathbb{SS}} MPIF(S) \quad (3)$$

being \mathbb{SS} , the complete search space, conformed by the set of all feasible combinations of interconnected facilities. As it was aforementioned, the root node is always selected to host the central facility.

Figure 2a illustrates a feasible solution $S_1 = \{0, A, D\}$ based on the graph depicted in Fig. 1 with threshold value $r = 10$. Selected facilities are highlighted with a gray background where, as it was described above, facility 0 always belongs to each feasible solution. In order to evaluate the objective function, it is required to compute the minimum sum of weights between each demand point and its closest facility. For example, the shortest path between demand point E and facilities in S_1 is 11 (path E-A). Similarly, the shortest path between G and S_1 is 6 (path G-F-A). For the sake of clarity, a dashed line indicates which facility serves each demand point.

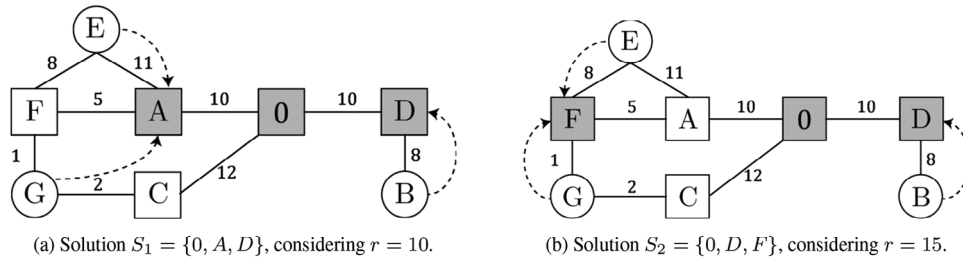


Fig. 2. Two possible solutions for the example depicted in Fig. 1, considering different values of r .

The evaluation of the objective function for S_1 is $MPIF(S_1) = w_{AE} + w_{AG} + w_{BD} = 11 + 6 + 8 = 25$. Notice that this solution verifies the threshold constraint r , since $d(0, A) = 10$ and $d(0, D) = 10$.

Departing from the same graph depicted in Fig. 1, we show in Fig. 2b the solution $S_2 = \{0, D, F\}$ with a different threshold value ($r = 15$). In this situation, the evaluation of S_2 is $MPIF(S_2) = w_{BD} + w_{GF} + w_{EF} = 8 + 1 + 8 = 17$. As expected, this solution also verifies the threshold constraint, that is, $d(0, D) = 10$ and $d(0, F) = 15$.

Figure 2a and b highlights the relevance of the threshold value r to the quality of the solution. In particular, increasing its value makes the problem less restrictive. Additionally, initializing the root node at 0 clearly limits the search, resulting in solutions where the central facility, 0 is not the closest to either of the demand points in some cases.

Note that Cherkesly et al. (2019) hold that the set of demand points, D , and the set of candidate facilities F , may be disjoint or may intersect. It is worth mentioning that, in the considered dataset, $D = F$ (we refer the reader to Section 4 for a detailed description of the dataset).

3. Variable neighborhood search

VNS methodology, originally proposed by Mladenović and Hansen (1997), is based on the idea of systematically changing the neighborhood structures in order to escape from a possible local optimum. Note that a solution S' in the neighborhood of S , denoted as $N(S)$, is obtained when a move operator is applied to perform a small perturbation to the solution S . The methodology relies on three main principles as stated in Hansen et al. (2017): (1) a local optimum with respect to one neighborhood structure is not necessarily a local optimum for another neighborhood structure, (2) a global optimum is a local optimum for all neighborhood structures, and (3) empirical evidence shows that for many problems, a large majority of local optima are relatively close to each other. Additionally, even in those problems in which local optima are far from each other, VNS is able to attain a good performance thanks to the neighborhood structure changes.

VNS starts from an initial solution and defines a maximum number of neighborhood structures to be explored. In general, a neighborhood structure $\mathcal{N}(S)$ of a solution S is defined as the set of solutions that can be obtained by performing a specific move operator over solution S . The algorithm then explores the neighborhoods considered from the first to the last one, using a local improvement phase to find a local optimum in each of the neighborhoods explored. If the local optimum found in a given neighborhood outperforms the best solution found so far, then it will

Algorithm 1. VNS(S, k_{\max}, δ)

```

1: for  $i = 1 \dots \delta$  do
2:    $k \leftarrow 1$ 
3:   while  $k \leq k_{\max}$  do
4:      $S' \leftarrow \text{Shake}(S, k)$ 
5:      $S'' \leftarrow \text{Improve}(S')$ 
6:     if  $\text{MPIF}(S'') < \text{MPIF}(S)$  then
7:        $S \leftarrow S''$ 
8:        $k \leftarrow 1$ 
9:     else
10:       $k \leftarrow k + 1$ 
11:     end if
12:   end while
13: end for
14: return  $S$ 

```

be updated, restarting the search from the first neighborhood. Otherwise, the algorithm continues the search by switching to the next neighborhood structure. This procedure is repeated until a certain stopping criterion is met, which is usually a limited computing time or a predefined number of repetitions.

The neighborhoods can be explored following different criteria. On the one hand, if they are randomly explored, then that results in a reduced VNS (Sánchez-Oro et al., 2014), where no local search is applied. On the other hand, a variable neighborhood descent (Duarte et al., 2018) is obtained when the neighborhoods are explored following a completely deterministic scheme. In this paper, we consider the basic VNS (BVNS) variant (Lozano-Osorio et al., 2020), which tries to balance intensification and diversification during the search by randomly selecting a solution in a neighborhood but locally improving it. It is worth mentioning that VNS is a metaheuristic which is in continuous evolution. For instance, Mladenović et al. (2019) proposed the less is more approach (LIMA), based on the BVNS, which relies on the idea that maintaining simple neighborhood structures allows the search for better quality solutions.

MPIF is solved using a BVNS approach in order to provide high-quality solutions in a reduced computing time. Details of the pseudocode are displayed in Algorithm 1.

Algorithm 1 receives three input parameters: a solution S that represents a feasible initial facility location (see Section 3.1), the maximum predefined neighborhood k_{\max} , and the allowed number of repetitions δ (stopping criterion). As it can be observed, VNS performs δ repetitions (steps 1–13). In each one, the method starts from the first neighborhood (step 2), and iterates until reaching the maximum predefined neighborhood k_{\max} (steps 3–12). Each repetition starts by randomly perturbing the incumbent solution S with the shake procedure described in Section 3.3 with the aim of exploring the current neighborhood (step 4). Then, the local improvement phase described in Section 3.2 is applied to find a local optimum in the neighborhood under exploration (step 5). Finally, the method selects the next neighborhood to be explored with the neighborhood change method. In particular, if the new local optimum S'' outperforms the incumbent solution S (step 6), then S'' becomes the incumbent solution (step 7), and the search starts again from the first neighborhood (step 8). Otherwise, the search continues in the next neighborhood (step 10). The method ends when

Algorithm 2. *Greedy*($G = (V, E)$)

```

1:  $S \leftarrow \{0\}$ 
2: while  $|S| \neq p$  do
3:    $CL \leftarrow \{v \in V \setminus S : d(v, S) \leq r\}$ 
4:    $v \leftarrow \arg \min_{c \in CL} MP\text{IF}(S \cup \{c\})$ 
5:    $S \leftarrow S \cup \{v\}$ 
6: end while
7: return  $S$ 

```

the maximum number of predefined repetitions is reached, returning the best solution found during the search (step 14).

3.1. Construction phase

Any VNS scheme requires an initial solution that can be generated either at random or by using a more elaborate procedure. In this work, we consider both strategies. Specifically, the random procedure, denoted as *Random*, starts by including the root node 0 in the partial solution under construction. Then, a list with those facilities whose distance to any other facility in the partial solution is lower than or equal to r is constructed (i.e., generating a set of interconnected facilities). More formally, let S be a partial solution and let $v \in F \setminus S$ be a candidate facility to be incorporated in S . We define the distance between v and S as follows:

$$d(v, S) = \min_{u \in S} d(v, u) \leq r. \quad (4)$$

It is worth mentioning that v can only be incorporated in S if and only if $d(v, S) \leq r$. Therefore, the list of candidates is constructed as follows:

$$CL \leftarrow \{v \in V \setminus S : d(v, S) \leq r\}. \quad (5)$$

From that list, a facility is selected at random, including it in the partial solution. This procedure ends when p facilities have been selected. As it is customary in p -median problems, demand points are assigned to the closest facility. Random procedures are extremely fast methods. However, they typically construct solutions with moderate quality. In order to partially overcome this drawback, we execute the random method for a determined number of repetitions, returning the best solution found.

The second constructive method, denoted as *Greedy*, is based on a greedy criterion. In this case, instead of selecting a node at random from the candidate list (as in the *Random* method), *Greedy* selects the node that minimizes Equation (2).

Algorithm 2 shows the pseudocode of the proposed greedy constructive procedure.

The algorithm starts by inserting the root node, 0, into the solution (step 1). The method then iterates until it generates a feasible solution, that is, by selecting $p-1$ additional interconnected facilities (steps 2–6). Therefore, in each iteration, it is necessary to determine the candidate facility to locate at a distance smaller than or equal to the maximum allowed distance r (step 3). Then, the

constructive procedure selects the candidate facility whose inclusion in the solution produces the minimum increase in the value of the objective function (step 4). Note that here the objective function is also considered as the greedy function that serves us as a criterion to make a good decision to locate interconnected facilities. Once a facility has been selected, it will be included in the solution (step 5). The method ends when p facilities are selected, returning a feasible solution S (step 7).

Since the constructive procedure considers the value of the objective function as a greedy criterion and this is a computationally demanding task, we propose an efficient evaluation of the objective function. As mentioned above, the greedy constructive procedure iteratively adds new facilities to the solution under construction, so the proposed optimization is designed to avoid performing a complete evaluation of the objective function, which originally involved a computational complexity of $O(|F| \cdot |D|)$.

The constructive method requires to identify the best node to locate a facility according to Equation (2) (ties are broken randomly). To do so, once a new facility is added to the partial solution, the minimum distance between all facilities to the demand points is stored in a memory structure. This cached information is used in the evaluation of each candidate facility, where it is necessary to evaluate the distance between the candidate facility and each demand point. The main idea is to traverse all demand points by leveraging the cached information and checking if this candidate facility improves the minimum distance to any demand point. During the traversal, the minimum distance is increased to obtain the actual value of the objective function.

Summarizing, considering a straightforward approach, the computation of the greedy function was originally performed with a complexity of $O(|F| \cdot |D|)$, but using the cached information, it can be reduced to $O(|D|)$.

Note that in Section 4 the performance of the proposed constructive procedures will be evaluated.

3.2. Improvement phase

The improvement phase in the VNS framework is in charge of generating a better local optima with respect to a certain neighborhood by means of intensification strategies. Our intention is to propose an effective and efficient method. Therefore, we propose a local search method specially designed for the MPIF with the aim of minimizing the computational effort but providing high-quality solutions.

Designing a local search for the MPIF is a challenge since a move (interchanging a selected facility with a nonselected facility) might violate the threshold constraint. More precisely, removing a facility may leave one or more facilities disconnected from the remaining ones, that is, their distance to other facilities could be larger than r . Let us illustrate this fact with an example. In Fig. 3a, we show a solution with six facilities, where we consider $p = 3$ (i.e., the solution is composed of facilities 0, A, and B) and $r = 10$. For the sake of clarity, we do not represent demand points (depicted with dotted lines in Fig. 3a). A move involving facility A disconnects facility B. Indeed, facility A cannot be interchanged with, say, for instance, facility E, since facility B remains unconnected (i.e., the minimum distance is still larger than 10). Therefore, the removal of facility A also involves the removal of facility B.

In order to overcome this situation, we propose a move which consists of removing not only the incumbent facility but also those facilities that make the solution unfeasible. After this removal,

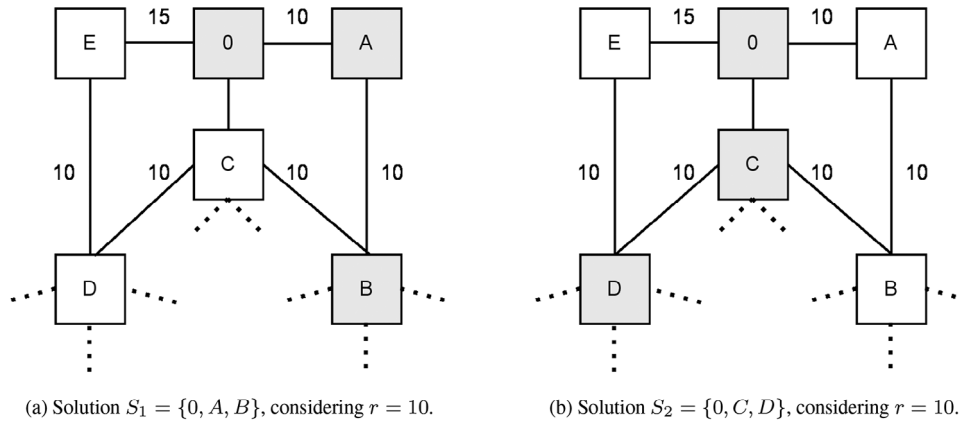


Fig. 3. Two possible solutions, considering different values of r .

these facilities are replaced with new facilities until a feasible solution is reached again. In Fig. 3(b) we show the solution after the move, i.e., removing facility A that forces the removing of facility B, and then, facilities C and D become part of the solution, making it feasible again.

It is important to remark that we are proposing a local search method. Therefore, a move is only performed if and only if it drives to an improvement in the objective function. As a consequence, new facilities are selected following the same greedy strategy as that presented in Section 3.1

Given a solution S and a facility $v \in S$, we define the move $Replace(S, v)$, which consists of replacing v (and those facilities that become disconnected), incorporating as many facilities as necessary to reach the feasibility. Note that the removal of a facility would force to remove others, producing a “cascade effect.”

The definition of move generates a neighborhood of solutions that can be reached by a single replacement move, which will be the neighborhood explored by the local search strategy. In particular, let us define $N_R(S)$ as the set of all solutions that can be reached by applying a single replace move to the solution S . More formally,

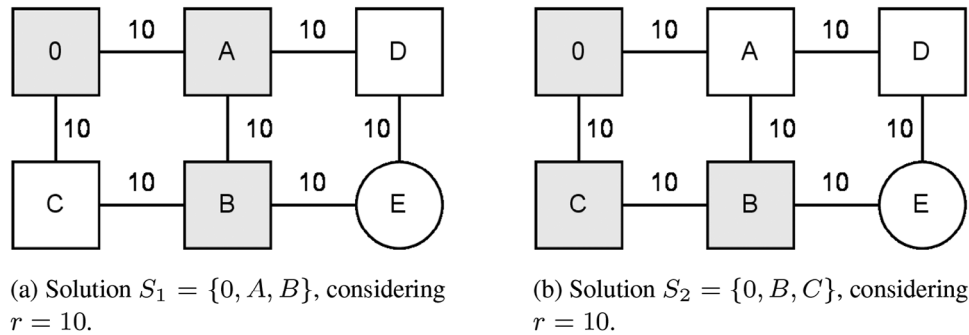
$$N_R(S) \leftarrow \{S' \leftarrow Replace(S, v) : \forall v \in S \setminus \{0\}\}. \tag{6}$$

There are two traditional improvement strategies: the best and the first improvement. On the one hand, the former traverses the complete neighborhood selecting the best neighbor solution found. On the other hand, the latter resorts to the first neighbor solution that results in an improvement. Both strategies end when any move does not drive to an improvement.

In Section 4, the comparison between the first and the best improvement approaches is shown for the MPIF.

3.3. Shaking phase

One of the main features of the VNS metaheuristic is the shaking phase, which is designed to escape from a possible local optimum as well as to increase the diversity of the search by means of a

Fig. 4. Two possible solutions, considering different values of r .**Algorithm 3.** *Shake*(S, k)

```

1:  $i \leftarrow 0$ 
2: while  $i < k$  do
3:    $L \leftarrow S \setminus \{0\}$ 
4:   if  $L \neq \emptyset$  then
5:      $v \leftarrow \text{Random}(L)$ 
6:      $I(v) \leftarrow \{c \in F \setminus (S \cup \{v\}) : d(w, (S \setminus \{w\}) \cup \{c\}) \leq r \forall w \in S\}$ 
7:      $u \leftarrow \text{Random}(I(v))$ 
8:      $S \leftarrow S \setminus \{v\}$ 
9:      $S \leftarrow S \cup \{u\}$ 
10:     $i \leftarrow i + 1$ 
11:     $L \leftarrow L \setminus \{v\}$ 
12:   end if
13: end while
15: return  $S$ 

```

perturbation of the solution. To that end, it is necessary to define a set of neighborhood structures, $\mathcal{N} = \{\mathcal{N}_1, \dots, \mathcal{N}_{k_{\max}}\}$, where \mathcal{N}_k , with $1 \leq k \leq k_{\max}$ maps a given solution S to a predefined neighborhood $\mathcal{N}_k(S)$. Note that the maximum neighborhood to be explored, k_{\max} , is an input parameter of the algorithm. Therefore, the neighborhood structure $\mathcal{N}_k(S)$ is defined as the set of solutions that can be obtained by replacing any k selected facilities (excluding the root node).

Shake procedure and local search must complement each other. Then, considering that the aforementioned improvement method might considerably modify the solution, we propose a shake method that produces a moderate perturbation of the corresponding solution. In addition, our intention is to design an efficient method, since the “cascade effect” produced by the local search could be highly time-consuming. Figure 4a depicts a solution $S_1 = \{0, A, B\}$, $p = 3$, and $r = 10$. Let us assume that we want to interchange facility A with any of the nonselected facilities (i.e., C and D). In this situation, the interchange between A and D produces an unfeasible solution, since facility B would be disconnected. However, the interchange with facility C produces a feasible solution (see Fig. 4b).

Algorithm 3 describes the pseudocode for the shake procedure. The first neighborhood to be explored is set in step 1. The algorithm performs at most k iterations (steps 2–13). In order to

detect if a move is feasible, we work with a copy of the incumbent solution in set L (step 3). Then, L is scanned in search of a feasible move if there are available facilities to be interchanged (steps 4–12). Specifically, a random facility is selected (step 5), obtaining a set $I(v)$ with those facilities that can be interchanged with v without disconnecting the remaining ones (step 6). The interchange is actually performed by selecting at random a new facility to be incorporated into the solution (step 7). Then, it is necessary to update the solution (steps 8 and 9), and finally, the next neighborhood to be explored (step 10). Finally, the perturbed solution is returned in step 14.

A naïve implementation of set $I(v)$ requires to ensure that the distance between each pair of facilities in the solution is smaller than or equal to r . However, this is a rather time-consuming approach, so we propose an optimization to reduce the associated computational time. Given a facility $v \in S$, we define the set of adjacent selected facilities as

$$N(v) = \{w \in S : d(v, w) \leq r\}. \quad (7)$$

Similarly, any nonselected facility $u \in F \setminus S$ belongs to $I(v)$ if and only if $N(v) \subseteq N(u)$, where

$$N(u) = \{w \in S : d(u, w) \leq r\}. \quad (8)$$

In other words, we only need to test whether the adjacent selected facilities of v are contained in the adjacent selected facilities of u or not. In that situation, the interchange does not disconnect the set of facilities. The effect of this strategy reduces the computational complexity of a straightforward implementation (a breadth-first search procedure) from $O(p^2)$ to $O(|N(v)|)$. Notice that this optimization can be considered as a preprocessing stage to reduce the computational effort of this phase.

The effect of the shake procedure together with the strategy proposed to construct the set $I(v)$ is evaluated in Section 4.

Summarizing, in Section 4 the performance of the whole proposal is evaluated: the constructive methods, as well as the effects of the acceleration strategy.

4. Computational results

This section reports the computational experiments to validate and test the efficiency and efficacy of the proposed VNS algorithm. All experiments were carried out on an AMD EPYC 7282 (2.8 GHz) with 16 GB RAM, and the proposed algorithm was implemented in Java 11 (build 11.0.11+9-Ubuntu-0ubuntu2.20.04, mixed mode, sharing). We would like to thank Cherkesly et al. (2019), the authors of the previous work, for kindly sending us their source code. This has helped us to provide a fair comparison by executing all algorithms under the same hardware environment also considering the same configuration as the one used in the original research.

We have used the set of instances introduced in the previous work (Cherkesly et al., 2019), which is based on the p -med instances derived from the well-known OR-Library.² Considering the original

²<http://people.brunel.ac.uk/~mastjjb/jeb/info.html>

Table 1
Summary of instances used in this work

	Number of instances	Number of nodes	Number of facilities	Distance
<i>p</i> -med instances	120	100–900	5–200	25–150
F-instances	108	500–3000	5–50	700–4000

set of 40 instances, Cherkesly et al. (2019) proposed a new set with 120 instances, where the set of facilities and demand points are the same. Therefore, instead of referring to “number of facilities” and “number of demand points,” we generally refer to “number of nodes.” Cherkesly et al. (2019) denote this set as *p*-med instances. The number of nodes ranges from 100 to 900, the number of facilities to be hosted ranges from 5 to 200, respectively, and the parameter *r* (threshold constraint) ranges from 25 to 150.

We propose a new set (denoted as F-instances) with large instances based on the *k*-median problem (Ahn et al., 1988) in order to provide large test cases with a tighter interconnection requirement. This set is divided into several subsets: F5, F10, F20, F30, F40, and F50; where the number close to the letter “F” indicates the number of facilities to be hosted. For each subset *FX*, we consider instances with 500, 1000, 1500, 2000, 2500, and 3000 nodes; and the *r* threshold constraint with 700, 2300, and 4000 units of distance. Then, there are 18 instances in each subset, resulting in a total of 108 instances to solve.

Table 1 summarizes the relevant information about these instances. Specifically, the first column indicates the name of the library where instances were selected, the second column remarks the total number of instances to be solved, the third column shows the minimum and maximum number of nodes, the fourth column contains the minimum and maximum number of facilities to host, and finally, the last column includes the minimum and maximum threshold distance considered to interconnect facilities. To facilitate future comparison, all instances and source codes are publicly available at <https://grafo.etsii.urjc.es/MPIF>.

The section is divided into two parts: preliminary experiments and final results. The former is focused on the selection of the best parameter settings. These experiments are performed on a representative subset of instances (46 out of 228 instances with different characteristics, which represent 20% of the instances) randomly selected. This selection allows us to analyze the behavior of the algorithm with unseen instances as well as to avoid overfitting. The final results compare the performance of our proposal with the state of the art, a metaheuristic proposed by Cherkesly et al. (2019) consisting of a hybrid construction followed by an iterative search. Furthermore, when it is possible, their mathematical models are used to solve problems to compare our proposal with the optimal value.

To facilitate comments made throughout the paper, all tables contain the following performance metrics: the average objective function value, Average O.F.; the average deviation with respect to the best solution in the experiment, Dev.(%); the number of times that the algorithm is able to attain the best value in the experiment (#Best), and, finally, the average execution time of the algorithm measured in seconds, Time (seconds). Some of the experiments have an additional column named “Average initial,” which indicates the average objective function value in which the algorithm starts. Notice that this value can only be set for the proposal but not for the previous works. Best results are highlighted with bold font.

Table 2
Comparison between one greedy and 400 random constructions

	Average O.F.	Dev.(%)	#Best	Time (seconds)
Random	1,133,752.57	25.07	1	0.55
Greedy	892,319.04	0.02	45	0.46

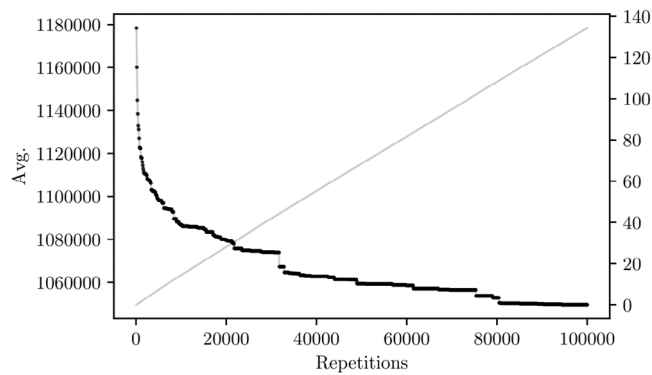


Fig. 5. Evolution of the quality of random constructions when increasing the number of repetitions.

4.1. Preliminary experiments

As introduced, the proposed VNS requires an initial solution that has been built using an approach based on greedy construction. Therefore, the first experiment is performed to evaluate the influence of using a greedy constructive procedure instead of random initial solutions. Table 2 shows the results obtained when considering one greedy construction and 400 random constructions, being the output of the best solution found along with the 400 constructions. The number of random constructions has been fixed with the aim of having a comparable computing time. These results highlight the relevance of a good constructive method to start from a promising region of the search space. In particular, the random constructive procedure is able to reach just one best solution, while the greedy procedure reaches the best solutions in the remaining 45 instances. Additionally, the deviation of the random procedure is drastically large, indicating that it is not even close to the solution found by the greedy constructive procedure.

It is important to analyze the evolution of the random constructive, with the aim of justifying the number of random constructions considered in the previous experiment. To that end, Fig. 5 shows the quality obtained by the random constructive when increasing the number of solutions generated (repetitions). Notice that, in the maximum value of 100,000 repetitions, the random constructive requires, on average, from 134.42 seconds to the average of 0.46 seconds of the greedy procedure.

It is worth mentioning that the random generation stagnates when reaching approximately 80,000 repetitions, making the quality of the solutions considerably worse than the quality of the greedy procedure, even with the maximum number of repetitions.

The next experiment is devoted to evaluate the influence of both the first and best improvement strategies in the context of the local search method. To this end, Table 3 summarizes the results

Table 3

Comparison between the first and the best improvement in the local search phase. Notice that both start from an average objective function value of 892,319.04 obtained by the constructive method (see Table 2)

	Average O.F.	Dev.(%)	#Best	Time (seconds)	# Iterations (Avg. \pm Std. Dev.)
First improvement	870,115.39	0.03	38	28.37	122.52 \pm 139.39
Best improvement	871,446.63	0.18	25	17.83	35.59 \pm 39.11

Table 4

Evaluation of the impact of parameters k_{\max} and δ in the final VNS algorithm

	0.1	0.2	0.3	0.4	0.5
100	342.57 / 0.11% / 30	450.84 / 0.04% / 36	342.57 / 0.11% / 30	514.03 / 0.02% / 38	539.09 / 0.03% / 40
200	415.69 / 0.08% / 30	491.87 / 0.04% / 37	415.69 / 0.08% / 30	560.10 / 0.02% / 38	593.09 / 0.03% / 40
300	461.14 / 0.08% / 33	515.78 / 0.04% / 37	461.14 / 0.08% / 33	594.14 / 0.02% / 39	618.66 / 0.02% / 42
400	477.08 / 0.08% / 34	533.93 / 0.04% / 37	477.08 / 0.08% / 34	615.21 / 0.02% / 39	626.61 / 0.02% / 42
500	488.99 / 0.08% / 34	551.98 / 0.03% / 37	488.99 / 0.08% / 34	621.48 / 0.02% / 39	634.57 / 0.02% / 42

obtained when solving the subset of 46 training instances used in the preliminary experiments. As we can observe, the first improvement strategy outperforms the best improvement strategy. It seems that the best improvement strategy gets stuck in a local optimum faster than the first improvement, thus exploring a narrower portion of the search space. This result can be partially explained when considering the computing time. As it is well documented in the related literature, the first improvement is usually faster than the best improvement because the first improvement is able to do much more iterations than the best improvement and, therefore, the best improvement gets stuck. However, in the set of instances considered, the first improvement requires, on average, 1.71 seconds, while the best improvement requires 1.10 seconds. Then, the first improvement has a better performance than the best improvement, obtaining the best result in 39 out of 46 training instances, which compares favorably to the best improvement (26 out of 46). Notice that the influence of the constructive procedure over the computing time is negligible. The last column of the table indicates the average and standard deviation with respect to the number of iterations that each strategy is able to perform. As expected, the first improvement performs a larger number of iterations than the best improvement, which may eventually lead the strategy to reach better results.

Having defined the constructive procedure and the local search method, it is necessary to tune the parameters of the proposed VNS algorithm. In particular, k_{\max} , the maximum neighborhood to be explored; and δ , the number of repetitions performed. Notice that both parameters must be tuned simultaneously since fixing one of them and then the other one does not guarantee choosing the best combination. In order to do so, we propose to test the combination of the values $k_{\max} = \{0.1p, 0.2p, 0.3p, 0.4p, 0.5p\}$, where each value depends on p to favor scalability and $\delta = \{100, 200, 300, 400, 500\}$ (we do not consider larger values, since this would result in a highly computationally demanding algorithm). These values are included in the final VNS algorithm, considering the greedy constructive procedure and the local search method for the first improvement. Table 4 shows the results obtained in this experiment.

Table 4 reports for each pair (k_{\max}, δ) the computing time in seconds, the average deviation, and the number of best solutions found in the experiment in the format *Time (seconds) / Dev (%)*

Table 5
Evaluation of the impact of parameters k_{\max} and δ in terms of effectiveness of the final VNS algorithm

	0.1	0.2	0.3	0.4	0.5
100	3.50 / 19.96 / 157.80	3.72 / 13.00 / 155.03	3.83 / 13.37 / 156.34	4.11 / 5.33 / 124.38	4.43 / 8.70 / 183.22
200	3.57 / 26.93 / 160.29	3.96 / 16.96 / 175.55	3.89 / 15.63 / 159.08	4.20 / 10.33 / 131.51	4.48 / 9.02 / 184.15
300	3.65 / 39.43 / 180.73	3.96 / 16.96 / 175.55	3.93 / 24.26 / 168.08	4.24 / 16.74 / 138.98	4.48 / 9.02 / 184.15
400	3.70 / 47.26 / 183.99	3.96 / 16.96 / 175.55	4.00 / 34.11 / 175.29	4.24 / 16.74 / 138.98	4.48 / 9.02 / 184.15
500	3.70 / 47.26 / 183.99	3.96 / 16.96 / 175.55	4.00 / 34.11 / 175.29	4.24 / 16.74 / 138.98	4.48 / 9.02 / 184.15

/ #Best. As expected, the combination that produces the best values in terms of deviation and the number of best solutions requires a larger computing time. However, it is necessary to find a balance between quality and performance. As can be observed, the larger the k_{\max} values, the better the outcomes. Nevertheless, it is necessary to reach a balance between effectiveness and efficacy. In order to do so, we have colored in gray the background of the most promising combinations, which are those that present high-quality solutions in reasonable computing time. In particular, $k_{\max} = 0.2$ and $\delta = 100$ have been selected, since they provide one of the best average deviation (0.04%) and a considerably large number of best solutions found (36 out of 46) while maintaining one of the most reduced computing time (450.84 seconds).

To perform a deeper analysis of the behavior of each variant along the considered repetitions, Table 5 shows, for each variant (cell of the table), three numbers in the format “A / B / C,” where A is the average number of repetitions where an improvement is found and B indicates, on average, the last repetitions in which an improvement is found. Additionally, in order to show how many seconds are effectively used for improving before stagnating, C indicates, on average, the computing time in which the last improvement was found. Notice that the average initial for CPLEX and ILS are not available, and they have been represented with a dash “-”.

The results obtained reinforce the idea of selecting $\delta = 100$ and $k_{\max} = 0.2$ since there are no significant improvements neither in the number of repetitions improving nor in the repetitions in which the last improvement is found. In terms of computing time, it can be seen that even varying the value of k_{\max} , the time to best is rather similar among all variants.

The next preliminary experiment is intended to evaluate the influence of the optimization strategy described at the end of Section 3.1 (i.e., efficient evaluation of the objective function). In order to do so, we execute a single repetition of the VNS ($\delta = 1$) with a straightforward and efficient evaluation of the objective function. We show in Fig. 6a time profile for both implementations referred to as “Efficient” and “Straightforward.” On the X -axis, we name each instance used in the preliminary experimentation with an ordinal number (from 0 to 45) and, on the Y -axis the associated computing time in logarithmic scale. As can be observed, the efficient implementation is more than one order of magnitude faster than the straightforward implementation. On average (considering the subset of 46 training instances), being the proposed efficient strategy, it is 52 times faster (27.13 vs. 1405.06 seconds).

The last experiment is devoted to show that MPIF is a highly constrained problem. In particular, the hypothesis is that the problem becomes more constrained when reducing the size of r since the search space is more difficult to explore due to new restrictions in the movements. In order to do so, we have conducted an experiment to analyze the improvement found by the local search and

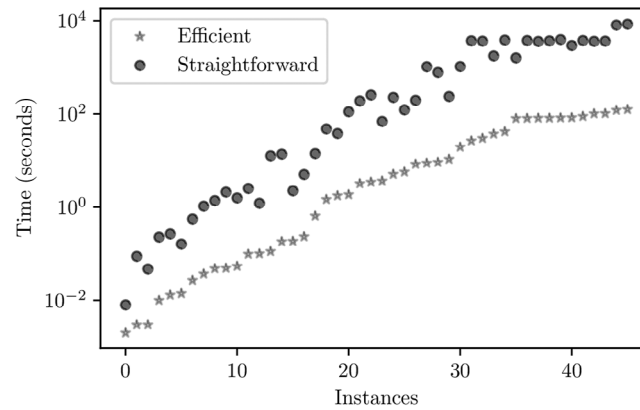


Fig. 6. Comparison between evaluation of the objective function (black) versus efficient evaluation of the objective function (gray).

Table 6

Analysis of the improvement found when decreasing the value of r , thus becoming a more constrained problem

	Greedy	LS	VNS	Gap Greedy - LS	Gap Greedy - VNS
$r = 150$	4,699.83	4,537.00	4,528.67	6.70%	6.98%
$r = 100$	4,796.33	4,593.33	4,554.00	9.35%	10.37%
$r = 80$	4,886.50	4,732.17	4,695.17	8.34%	9.47%
$r = 50$	10,013.33	9,884.33	9,875.33	3.92%	4.50%
$r = 25$	14,767.67	14,764.00	14,763.00	0.05%	0.07%

VNS with respect to the initial solution found by the greedy constructive procedure. The metric selected to show the improvement is the gap between the initial and final solutions, evaluated as $\frac{|Initial - Final|}{Final}$, where $Final$ is the best value found and $Initial$ is the objective function value of the initial solution. The larger the value, the better the improvement. Table 6 shows the results obtained in this experiment.

As it can be derived from the results, VNS is able to consistently find better results than the local search isolated even in the most constrained instances (those with $r = 25$). The results confirm the initial hypothesis, being both local search and VNS unable to find significant improvements with respect to the initial solution. However, when considering larger values of r , the gap in improvement reaches approximately 10%. Analyzing these results, we can conclude that MPIF is a highly constrained problem that becomes more difficult when decreasing the size of r .

In order to analyze if there are statistically significant differences between the results in each stage, we have applied the pairwise Wilcoxon test comparing Greedy with LS, Greedy versus VNS, and LS versus VNS, all of them resulting in a p -value smaller than 0.001, indicating that there are statistically significant differences among them.

Finally, since VNS may benefit from initial diverse solutions, we have conducted an experiment replacing the initial greedy solution with a randomly generated one. The results show that using a random initial solution results in consistently worse solutions, highlighting the contribution of the greedy constructive procedure.

Table 7

Comparison of CPLEX, VNS, and ILS when considering the 75 instances from the OR-Library where the optimum value is reached by CPLEX

	Average initial	Average O.F.	Dev.(%)	#Opt	Time (seconds)
CPLEX	–	4778.80	0.00	75	50.08
VNS	4847.55	4781.11	0.07	61	32.78
ILS	–	4788.56	0.34	35	10.09

4.2. Final results

The goal of this section is to test the performance of the proposed VNS by comparing it with the best method found in the state-of-the-art. In particular, Cherkesly et al. (2019) proposed an ILS and an integer linear programming (ILP) formulation which is able to optimally solve small instances. For the sake of clarity, the experiments have been divided into three well-differenced parts. The first one is devoted to evaluate whether the heuristic approaches (ILS and VNS) are able to reach optimal values when ILP (originally implemented in CPLEX 12.7.1) certifies the optimum within the prescribed one-hour time limit for the p -med instances. The second part compares the performance of the proposed VNS with ILS when again considering the p -med instances in which CPLEX is unable to reach the optimum value within the prescribed one-hour time limit. Finally, the last part compares ILS and VNS in the new set of instances larger with a tighter interconnection requirement than the p -med instances, the F-instances, described at the beginning of this section.

Therefore, the first experiment considers the subset of 75 out of 120 instances derived from the OR-Library and widely known as the p -med instances, where CPLEX is able to reach the optimal value. Although in the OR-Library there are 120 instances in total, the exact model is unable to solve 45 of them, which is the main reason why in the following experiment shown in Table 7 only 75 instances are included. Therefore, Table 7 summarizes the results obtained by comparing CPLEX, VNS, and ILS in those instances. Note that, in column 4, instead of the number of times that the algorithm is able to get the best value (#Best), we have shown the number of times that the algorithm is able to get the optimum value (#Opt). Note that, Cherkesly et al. (2019) proposed two mathematical formulations to solve the MPIF, but as they hold, model MPIF-2 outperforms model MPIF-3 and, therefore, we have used model MPIF-2.

In view of the results and as expected, CPLEX provides all optimum values, reaching a deviation of 0.00% and finding 75 out of 120 optimal solutions. However, by analyzing the performance of VNS and ILS, it can be affirmed that both algorithms are able to reach near-optimal solutions in all cases, providing VNS with a better deviation than ILS (0.07% vs. 0.34%). Furthermore, the VNS algorithm is able to attain 61 out of 75 optimal solutions, meanwhile the ILS algorithm reaches 35 out of 75 optimal solutions, respectively.

It is worth mentioning that the apparently large value of the VNS computing time is due to the parameter configuration described in the previous section (i.e., $k_{\max} = 0.2$ and $\delta = 100$). However, for these small instances, the VNS requires considerably less computing time to find the best solution. Specifically, the time-to-best for these instances is 7.18 seconds, which is considerably smaller than 32.78 seconds.

Table 8
Comparison of VNS and ILS when considering all instances from the OR-Library

		Average initial	Average O.F.	Dev.(%)	#Best	Time (seconds)
$r = 150$ (6)	VNS	4574.67	4528.67	0.00	6	1.34
	ILS	–	4528.67	0.00	6	0.12
$r = 100$ (9)	VNS	4549.00	4484.56	0.00	9	5.20
	ILS	–	4490.89	0.13	5	0.31
$r = 80$ (40)	VNS	5720.75	5666.73	0.00	39	36.67
	ILS	–	5674.93	0.27	20	27.82
$r = 50$ (34)	VNS	5903.18	5851.38	0.00	34	42.63
	ILS	–	5854.88	0.13	19	30.23
$r = 25$ (31)	VNS	6151.10	6077.48	0.01	29	46.16
	ILS	–	6083.29	0.24	15	20.22
Average (120)	VNS	5738.43	5679.59	0.00	117	36.68
	ILS	–	5685.29	0.20	65	23.09

Now, we compare the performance of ILS and VNS when considering the complete set of p -med instances. We have also included the 75 instances where the ILP is able to obtain the optimum value to facilitate direct comparison between both metaheuristics. Table 8 summarizes the results obtained, where each main row contains the averaged results per r -threshold over the number of instances reported in parentheses.

It is worth mentioning that the VNS algorithm outperforms the results obtained by the ILS, the average value of the objective function (5679.59 vs. 5685.29) and the average deviation (0.00% vs. 0.20%) as can be observed in Table 8. Furthermore, in terms of computational time, VNS consumes 36.68 seconds versus 23.09 seconds in the ILS. Analyzing in detail the best solutions found, it is important to emphasize that in 55 out of 120 instances, better results are obtained using the VNS; in three out of 120 instances, the VNS is not able to reach the best-known solution; and, finally, in 62 out of 120 instances, both algorithms match the best solution. Therefore, we can hold that results obtained using VNS are significantly better than the results obtained using ILS. We confirm this fact by conducting a pairwise Wilcoxon signed rank test in which the null hypothesis tests if the difference between the paired results obtained by each algorithm (VNS and ILS) is zero. A p -value < 0.001 has been obtained; therefore, we can ensure that there are significant differences between results.

The last experiment considers the new set of F-instances, which are larger and more complex than the ones in the OR-Library, resulting in a more challenging set of instances for comparing metaheuristic procedures. Table 9 shows the results obtained when comparing again VNS and ILS in this set of instances. As it can be seen, we have grouped the instances according to the number of facilities to be hosted.

In light of the results of Table 9, it can be affirmed that VNS is more competitive than ILS in all F-instances if we focus on the average values of the objective function and, therefore, the average deviation. Furthermore, the number of best solutions found in VNS always outperforms ILS, obtaining remarkable results. However, it is fair to mention that ILS is faster than VNS when considering the smallest subsets (F5 and F10), even though this happens only in those small instances. Nonetheless, when considering all F-instances, on average, the computing time consumed by VNS

Table 9
Comparison of VNS and ILS when considering all F-instances.

F-instances		Average initial	Average O.F.	Dev.(%)	#Best	Time (seconds)
F5 (18)	VNS	4,360,406.11	4,294,389.94	0.09	15	18.46
	ILS	–	4,312,042.33	0.51	10	3.34
F10 (18)	VNS	3,107,332.22	3,021,074.00	0.07	15	48.30
	ILS	–	3,040,697.06	1.00	8	21.55
F20 (18)	VNS	2,071,252.67	2,009,032.89	0.12	15	72.93
	ILS	–	2,018,848.28	0.88	5	81.45
F30 (18)	VNS	1,573,030.11	1,520,469.78	0.07	16	84.14
	ILS	–	1,532,471.50	1.26	2	157.36
F40 (18)	VNS	1,293,950.28	1,239,921.61	0.03	16	95.67
	ILS	–	1,258,758.83	1.55	2	253.98
F50 (18)	VNS	1,101,744.22	1,054,200.28	0.17	13	108.45
	ILS	–	1,060,101.61	0.56	5	379.39
Average (108)	VNS	2,251,285.94	2,189,848.08	0.09	90	71.33
	ILS	–	2,203,819.94	0.96	32	149.51

is more than two times faster than ILS, which confirms the scalability of the proposal. Moreover, VNS is able to find the best solution in 90 instances, while ILS in 32 instances, and there are 14 instances in which both methods attain the same solution.

To conclude the analysis over the set F-instances, a pairwise Wilcoxon signed rank test is conducted in order to validate if there are significant differences between the results obtained by both algorithms. As the p -value < 0.001 , we can confirm that there are statistically significant differences between the VNS and ILS algorithms. Therefore, it can be confirmed that VNS outperforms ILS. Then, VNS emerges as the most competitive algorithm for solving the MPIF.

5. Conclusions

In this paper, the p -median location problem with interconnected facilities is addressed. This problem appears in many real-world situations in which a central facility must be close to other facilities and, additionally, all the facilities must be close to the other ones in order to guarantee connectivity among them. Many real applications fit into this model, such as the location of forest fire stations, alarms, movement detectors, cordless audio devices, or radio frequency identifications, among others.

To tackle the considered problem, a VNS algorithm is designed. The obtained results prove the superiority of the proposal when compared with the state-of-the-art algorithm. The efficient evaluation of the objective function considerably reduces the computational effort. Additionally, we propose a new move in the shake procedure that maintains the feasibility of the incumbent solution and it considerably reduces the search space.

In future research, it would be interesting to study new variants of this problem following a similar approach (i.e., interconnected facilities) to evaluate its potential. One possibility is to consider that the root node is not determined in advance, contrary to the variant proposed by

Cherkesly et al. (2019) and, therefore, the one addressed in this paper. We do think that allowing the central facility to be located in a different node not compulsorily in the root node, will allow more flexibility in the model. Another alternative is to remove the cardinality constraint (not forcing a specific value of p) but consider costs associated with opening facilities.

Acknowledgments

The authors wish to express their gratitude to M. Cherkesly, M. Landete, and G. Laporte for their collaboration in the provision of their executable code to perform a fair comparison.

J. Sánchez-Oro, A. Duarte, and I. Lozano-Osorio acknowledge the support from the Spanish Ministry of “Ciencia, Innovación y Universidades” (MCIN/AEI/10.13039/501100011033/FEDER, UE) under grant ref. PID2021-126605NB-I00 and PID2021-125709OA-C22.

A.D. López-Sánchez acknowledges support from the Spanish Ministry of “Economía, Industria y Competitividad” through Projects PID2019-104263RB-C41, and from Junta de Andalucía, FEDER-UPO Research & Development Call, reference number UPO-1263769.

References

- Ahn, S., Cooper, C., Cornuéjols, G., Frieze, A., 1988. Probabilistic analysis of a relaxation for the k -median problem. *Mathematics of Operations Research* 13, 1–31.
- Cherkesly, M., Landete, M., Laporte, G., 2019. Median and covering location problems with interconnected facilities. *Computers & Operations Research* 107, 1–18.
- Duarte, A., Sánchez-Oro, J., Mladenović, N., Todosijević, R., 2018. Variable neighborhood descent. In Martí, R., Pardalos, P., Resende, M. (eds) *Handbook of Heuristics*. Springer, Cham, pp. 341–367.
- Feo, T., Resende, M., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 109–133.
- García-López, F., Melián-Batista, B., Moreno-Pérez, J., Moreno-Vega, J., 2002. The parallel variable neighborhood search for the p -median problem. *Journal of Heuristics* 8, 375–388.
- Hakimi, S., 1964. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research* 12, 450–459.
- Hakimi, S., 1965. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research* 13, 462–475.
- Hansen, P., Mladenović, N., 1997. Variable neighborhood search for the p -median. *Location Science* 5, 207–226.
- Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S., 2017. Variable neighborhood search: basics and variants. *EURO Journal on Computational Optimization* 5, 423–454.
- Herrán, A., Colmenar, J., Martí, R., Duarte, A., 2018. A parallel variable neighborhood search approach for the obnoxious p -median problem. *International Transactions in Operational Research* 27, 336–360.
- Lin, G., Guan, J., 2018. A hybrid binary particle swarm optimization for the obnoxious p -median problem. *Information Sciences* 425, 1–17.
- Lourenço, H., Martin, O., Stutzle, T., 2003. Iterated local search. *International Series in Operations Research & Management Science* 57, 320–353.
- Lozano-Osorio, I., Sánchez-Oro, J., Rodríguez-García, M., Duarte, A., 2020. Optimizing computer networks communication with the band collocation problem: a variable neighborhood search approach. *Electronics* 9, 1860.
- Mladenović, N., Hansen, P., 1997. Variable neighborhood search. *Computers & Operations Research* 24, 1097–1100.
- Mladenović, N., Alkandari, A., Pei, J., Todosijević, R., Pardalos, P., 2019. Less is more approach: basic variable neighborhood search for the obnoxious p -median problem. *International Transactions in Operational Research* 27, 480–493.
- Mousavi, S., Bhambar, S., England, M., 2023. An iterated greedy algorithm with variable reconstruction size for the obnoxious p -median problem. *International Transactions in Operational Research*.

- Reese, J., 2006. Solution methods for the p -median problem: an annotated bibliography. *Networks* 48, 125–142.
- Sánchez-Oro, J., Pantrigo, J., Duarte, A., 2014. Combining intensification and diversification strategies in VNS. An application to the vertex separation problem. *Computers & Operations Research* 52, 209–219.
- Srinidhi, N., Kumar, S., Venugopal, K., 2019. Network optimizations in the Internet of Things: a review. *Engineering Science and Technology, An International Journal* 22, 1–21.
- Tuba, E., Strumberger, I., Bacanin, N., Tuba, M., 2018. *Bare bones fireworks algorithm for capacitated p -median problem*. Lecture Notes in Computer Science, 10941, 283–291.