Computers and Operations Research 104 (2019) 295–303

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

GRASP and VNS for solving the *p*-next center problem

A.D. López-Sánchez [a,*], J. Sánchez-Oro [b], A.G. Hernández-Díaz [a]

[a] Pablo de Olavide University, Ctra. de Utrera km 1., Sevilla 41013, Spain
[b] Rey Juan Carlos University, Tulipán s/n., Móstoles, Madrid 28933, Spain

ARTICLE INFO

Article history:
Received 18 December 2017
Revised 17 December 2018
Accepted 18 December 2018
Available online 21 December 2018

Keywords:
Discrete location
p-center problem
p-next center problem
GRASP
VNS

ABSTRACT

This paper presents two metaheuristic algorithms for the solution of the *p*-next center problem: a Greedy Randomized Adaptive Search Procedure and a Variable Neighborhood Search algorithm, that will be subsequently hybridized. The *p*-next center problem is a variation of the *p*-center problem, which consists of locating p out of n centers and assigning them to users in order to minimize the maximum, over all users, of the distance of each user to its corresponding center plus the distance between this center to its closest alternative center. This problem emerges from the need to reach a secondary help center in the case of a natural disaster, when the closest center may become unavailable.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

The interest of this paper is motivated by very common situations in humanitarian logistics in which an unexpected incident can occur in any of the centers. When a center remains disabled, it is not able to provide help and all users are forced to be reassigned to another center, which is usually the closest center to that which has become unavailable.

When a disaster strikes, whatever the nature of the disaster (natural or caused by humans), any person injured or affected heads towards emergency assistance centers, such as hospitals, refugee camps, and rescue centers. For instance, in war situations, multitudes of people need access to hospitals and may need to escape from their own country to be hosted in refugee camps. Additionally, in natural disasters, such as earthquakes, tsunamis and hurricanes, rescue centers are set up to serve the affected population. In real situations, a person naturally chooses to go to the nearest emergency center, but it could be that the closest center is no longer available for any of multiple reasons (including overcrowding and running out of supplies, among others). In this case, the best alternative for that person is to find the next closest center. This problem, which is a variation of the *p*-center problem (*p*CP), has been named the *p*-next center problem (*p*NCP).

The *p*CP is one of the most frequently studied location problems and it can be formally stated as follows. Let $G = (V, E)$ be a complete graph, where $V = \{1, \ldots, n\}$ is the set of vertices that represents potential locations to host centers ($|V| = n$), and $E = \{(i, j) : i, j \in V, i \neq j\}$ is the edge set, where each edge (i, j) represents a path between vertices i and j. For each pair $(i, j) \in E$, let $d_{ij} \geq 0$ be the length of the shortest path that connects locations i and j which must satisfy $d_{ii} = 0$, for all $i \in V$, $d_{ij} > 0$, for all $i, j \in V$, $i \neq j$, and $d_{ij} \leq d_{ik} + d_{kj}$, $i, j, k \in V$. The objective in the *p*CP is to select a subset $P \subset V$ of cardinality p that minimizes the maximum of the distances between users and their closest centers in P. In other words, the aim is to locate p out of n centers while minimizing the maximum distance between a user and a center. Since each vertex represents a center located in a certain population, it is assumed that all the users living in that population can be represented by that vertex. The first paper on the *p*CP was published by Hakimi (1964) where the problem was defined and solved using graphical methods. A few years later, Minieka (1970) provides an iterative set covering based exact method to solve the *p*CP. Since then, a multitude of research studies have addressed the *p*CP that make use of exact or heuristic methods. Daskin (1995) proposes the first mixed integer programming formulation for the pCP. The author also provides an exact algorithm based on the idea of Minieka (1970) integrated by a bi-section method. Details on the *p*CP (a formulation and two different approaches: exact and heuristic) can be found in Elloumi et al. (2004), who present an integer linear programming formulation with a polynomial number of variables and constraints and include the linear programming relaxation to provide a lower bound; and Mladenović et al. (2003) who address the *p*CP heuristically implementing a Variable Neighborhood Search and two Tabu Search heuristics. Recently, Calik and Tansel (2013) proposed a new integer programming formulation. Davidovic et al. (2011) and

* Corresponding author.
E-mail address: adlopsan@upo.es (A.D. López-Sánchez).

https://doi.org/10.1016/j.cor.2018.12.017
0305-0548/© 2018 Elsevier Ltd. All rights reserved.

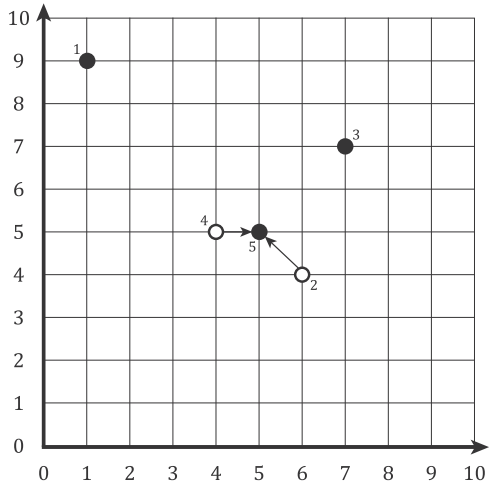

Fig. 1. Optimal location for the *p*-center problem.

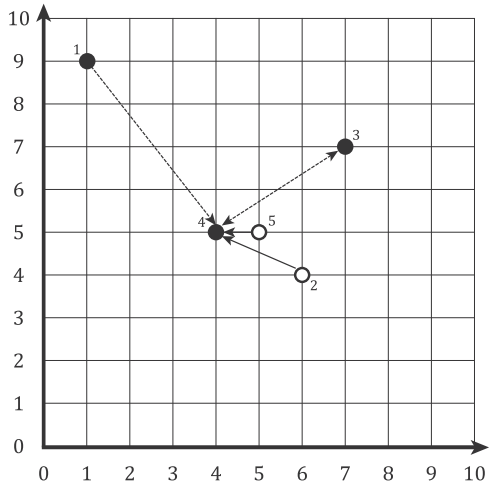

Fig. 2. Optimal location for the *p*-next center problem.

Irawan et al. (2016) solve the *p*CP with new methodologies, with methods based on bee colony and hybrid metaheuristics, respectively. We also refer the reader to Yin et al. (2017) and Martínez-Merino et al. (2017) since they solve the *p*CP using two methodologies similar to those implemented in this paper.

As introduced above, the *p*NCP is an extension of the *p*CP, and is more realistic in many actual situations. The *p*NCP consists of locating p out of n centers in order to minimize the maximum distance between users and their closest center (referred as the *reference center*) plus the distance from this reference center to its closest center (also known as the *backup center*). Thus, the *p*NCP can be formally defined as follows:

$$\min_{\substack{P\subset V\\ |P|=p}} \max_{i\in V} \left\{ \min_{j\in P}\{d_{ij}\} + \min_{\substack{k\in P\\ k\neq j'\in \arg\min_{j\in P}\{d_{ij}\}}} \{d_{j'k}\} \right\}$$

In order to graphically illustrate the difference between the *p*CP and the *p*NCP, a simple example is included in Figs. 1 and 2, where the optimal locations for the *p*CP and the *p*NCP are represented, respectively. Consider the problem defined by $n = 5$ with Euclidean distances and $p = 3$. The optimal solution value for the *p*CP is 1.41 units and centers are located at nodes 1, 3 and 5. The solution value, 1.41 units, is calculated as the maximum distance between a user and the reference center, which, in this case, is the length from node 2 to node 5. On the other hand, the optimal solution value for the *p*NCP is 5.00 units and centers are located at nodes 1, 3, and 4. The solution value, 5.00 units, is calculated as the maximum distance between each user and its reference center plus the distance of this center to its closest center, which, in this case, is the length from node 1 to node 1 since it is a reference center plus the length from node 1 to node 4.

As previously explained, when all centers are available, then users go to their closest reference center (*p*CP). In Fig. 1, users from center 1, 3 and 5 will stay in these centers since they are reference centers and users from 2 and 4 will go to center 5 since it is their closest reference center. However, as mentioned earlier, it could be that when users arrive at their reference center, it is not available. In such a case, users will go to the closest backup center (*p*NCP) and this situation is represented in Fig. 2. In that example, users from centers 1, 3, and 4 will stay in these centers since they are reference centers but, in the case that one of them is no longer available, they will go to their closest reference center (a backup center). The backup center of centers 1, 2, 3 and 5 is reference center 4 and the backup center of 4 is center 3. Although these two problems are very similar, the optimal solution values for the *p*CP and for the *p*NCP are substantially different since the reference centers are located on different vertices.

The *p*NCP was first introduced by Albareda-Sambola et al. (2015). The authors proved that this discrete optimization problem is NP-hard. They proposed and compared several different linear integer programming formulations of *p*NCP: a three-index formulation using path variables; a two-index formulation; and a formulation using covering variables. All formulations were implemented in Xpress Optimizer and the experiments were limited to 2 h of CPU time. Albareda-Sambola et al. (2015) were able to solve a set of instances of up to 50 nodes with a reasonable computational effort. To the best of our knowledge, this was the first and only attempt to address the *p*NCP.

In this paper, a different approach is proposed, and the limitations found in Albareda-Sambola et al. (2015) are taken into account. Various metaheuristics are designed specifically to solve the *p*NCP. Thus our goal is now to produce optimal or near-optimal solutions in a reasonable amount of time. It is well-known that when the size of the problem increases considerably, exact algorithms can remain unable to find the optimal solution due to their high cost in terms of computing time or lack of available memory. The difficulty of solving certain problems using exact algorithms leads us to using heuristic algorithms. To this end, the first algorithm implemented is a Greedy Randomized Adaptive Search Procedure and the second algorithm designed is a Variable Neighborhood Search methodology. Finally, the methodologies above are then hybridized in order to attain even better results. These metaheuristics are compared among them and against the optimal solution provided by Albareda-Sambola et al. (2015) when it is known or, otherwise, against the best-known solution. We prove their usefulness in solving the *p*NCP while ensuring high-quality solutions in reasonable computational times.

The rest of this paper is organized as follows. Section 2 describes the metaheuristics implemented to solve the problem under consideration. Section 3 presents the computational experiments performed to test the quality of the proposed methods. Finally, Section 4 summarizes the paper and discusses future work.

2. Algorithms

In this section the algorithms proposed for solving the *p*NCP are presented. The first methodology detailed in Section 2.1 is a multi-start algorithm known as the Greedy Randomized Adaptive Search Procedure (GRASP), while the second methodology presented in Section 2.2 is a trajectory-based algorithm named Variable Neighborhood Search (VNS).

GRASP uses a set of solutions that are subsequently improved in order to obtain high-quality solutions and this algorithm strengthens the diversification of the solutions. In contrast, the VNS algorithm starts from a single solution in order to attain a better quality solution and, in this case, this algorithm encourages the intensification of the solution.

2.1. Greedy randomized adaptive search procedure

The GRASP is a methodology originally proposed by Feo and Resende (1989) but it was formally introduced several years later by Feo et al. (1994). We refer the reader to Feo and Resende (1995); Quintana et al. (2016); Resende and Ribeiro (2010) for the analysis of recent surveys and successful applications of GRASP. This multi-start algorithm consists of a two-phase process: a randomized construction phase using a greedy function; and an improvement phase to reach a local optimum. These two stages are then repeated until a certain termination criterion is met. The most interesting feature of this methodology is the combination of greediness and randomness in the generation of solutions by means of restricted candidate lists (*RCL*) of variable sizes that control both features. Note that the greediness/randomness of the method is controlled by a parameter denoted by α. Note that, if $\alpha = 0$, then the method becomes completely greedy (it only considers the most promising element to be included in the solution under construction), and if $\alpha = 1$, then all elements are included in the *RCL*, resulting in a totally random method. For comprehensive literature reviews on GRASP algorithms, Festa and Resende (2008, 2009) are recommended, where algorithmic aspects were surveyed and applications were included into a variety of combinatorial optimization problems. More specifically, in relation with the *p*CP, we suggest Yin et al. (2017) who included a Path Relinking in a GRASP framework. The resulting algorithm is competitive with the state-of-the-art algorithms in terms of solution quality and computational efficiency.

Hereinafter, specific details of the proposed GRASP algorithm will be described. Algorithm 1 presents the constructive stage of

Algorithm 1 Construct($G = (V, E)$, α).

1: $P \leftarrow \emptyset$
2: $\{v, u\} \leftarrow Select(V)$
3: $P \leftarrow P \cup \{v, u\}$
4: $CL \leftarrow V \setminus \{v, u\}$
5: **while** $|P| \neq p$ **do**
6: $\quad g_{\min} = \min_{v \in CL} g(P, v)$
7: $\quad g_{\max} = \max_{v \in CL} g(P, v)$
8: $\quad th \leftarrow g_{\min} + \alpha(g_{\max} - g_{\min})$
9: $\quad RCL \leftarrow \{v \in CL : g(P, v) \leq th\}$
10: $\quad v' \leftarrow SelectRandom(RCL)$
11: $\quad P \leftarrow P \cup \{v'\}$
12: $\quad CL \leftarrow CL \setminus \{v'\}$
13: **end while**
14: **return** P

the GRASP algorithm for solving the *p*NCP.

It must be borne in mind that any solution for the *p*NCP is represented as a set P of p selected centers to which users will be assigned. The input for the method is comprised of the input graph $G = (V, E)$, and the parameter α. The constructive phase of the GRASP method proposed in this work starts from an empty solution P (step 1). The method then selects the first two centers $\{v, u\}$ to be added to the solution (steps 2 and 3), thereby constructing a list of candidate centers to be added, initially $CL = V \setminus \{v, u\}$ (step 4). The first two nodes are selected in two different ways, and these are denoted as C1 and C2. The first strategy for the selection of the two first nodes, C1, randomly selects the first two centers. On the other hand, the second strategy for the selection of the two first nodes, C2, evaluates all the vertices in G and selects the first vertex v as that which maximizes the following expression: $\max_{v \in N} \min_{u \in N} d_{vu}$. The second center selected is therefore the closest center to v.

Once the first two centers have been selected, the method iterates until the number of centers assigned to the solution under construction is equal to p (steps 5–13). At each iteration of the construction, all the vertices in $CL = V \setminus P$ are evaluated with a greedy function defined as: $g(P, v) = \max_{i \in V} \left\{ \min_{j \in P \cup \{v\}} \{d_{ij}\} + \min_{\substack{k \in P \cup \{v\} \\ k \neq j' \in \arg\min_{j \in P \cup \{v\}} \{d_{ij}\}}} \{d_{j'k}\} \right\}$ which evaluates the objective function value when adding the vertex v to the solution under construction, and maintains the best and worst values, $g_{\min}$ and $g_{\max}$, respectively (steps 6 and 7). These values are considered for the calculation of a threshold, denoted by th (step 8), which is useful for creating the restricted candidate list, *RCL* (step 9). The *RCL* contains the most promising candidate vertices to be added (i.e., those that yield the smallest increase of the objective function, taking into account the previously computed threshold). Finally, the method randomly selects a vertex v' from the *RCL* (step 10), and adds it to the solution P (step 11). The *CL* is then updated by removing the selected vertex (step 12). The method ends by returning the constructed solution when p centers have been selected.

The second stage of the GRASP is designed to obtain a local optimum with respect to a predefined neighborhood of the constructed solution. Given a solution P, the neighborhood $N_1(P)$ is defined as the set of solutions that can be obtained by exchanging a selected center with any non-selected center. In mathematical terms,

$$N_1(P) \leftarrow \{(P \setminus \{u\}) \cup \{v\} : u \in P, v \in V \setminus P\}$$

The proposed local search method follows a first-improvement strategy. In particular, the method visits all neighbor solutions by following a random order, and replaces the incumbent solution with the first neighbor solution that has a better objective function value; the search is subsequently restarted with the neighborhood of the updated solution. The search stops when a local optimum is reached (i.e., no better solution can be found in the neighborhood).

The construction and improvement phases are executed in the GRASP methodology until a stopping criterion is met, which is, in our case, the number of solutions generated. Specifically, the proposed GRASP is executed 100 times, which means, 100 initial solutions are built that are subsequently improved upon. The algorithm returns the best solution generated. Algorithm 2 depicts the

Algorithm 2 GRASP ($G = (V, E)$, α, *nsol*).

1: $P_b \leftarrow \emptyset$, $f(P_b) \leftarrow \infty$
2: **for** $i = 1 \ldots nsol$ **do**
3: $\quad P \leftarrow Construct(G = (V, E), \alpha)$
4: $\quad P' \leftarrow LocalSearch(P)$
5: $\quad$ **if** $f(P') < f(P_b)$ **then**
6: $\quad\quad P_b \leftarrow P'$;
7: $\quad$ **end if**
8: **end for**
9: **return** P_b

pseudo-code for the GRASP algorithm proposed in this work.

This algorithm requires the complete graph G and two parameters: α, which controls the greediness/randomness of the construc-

tive procedure; and *nsol*, which indicates the number of solutions generated. For each execution, the algorithm constructs a new solution with the constructive procedure (step 3) and then improves it with the local search method (step 4). Finally, if the improved solution P' is better than the best solution found so far, P_b, then it is updated. The method stops when *nsol* solutions have been generated, and returns the best solution found during the search, P_b.

2.2. Variable neighborhood search

VNS algorithms were originally proposed by Mladenović and Hansen (1997). VNS relies on the idea of systematic changes of neighborhood structures. The adaptability of the methodology has resulted in several variants in recent years (see Hansen et al. (2017) for a recent survey on the methodology), which has led to several successful applications for a variety of difficult optimization problems, such as those in Duarte et al. (2016) and Sánchez-Oro et al. (2015). Furthermore, other location problems have been addressed using VNS methodologies: one interesting problem was addressed by Martínez-Merino et al. (2017), where the probabilistic p-center problem that seeks to minimize the expected maximum distance between any vertex and its center was solved.

In this paper, we focus on the Basic VNS (BVNS) algorithm, which combines both stochastic and deterministic changes of neighborhood in order to obtain high-quality solutions. The neighborhoods have been defined in a similar way to the neighborhood N_1. Therefore, each neighborhood, N_k (with $k \leq p$), is defined as the set of solutions that can be reached by performing k exchanges between selected and non-selected centers. Note that p different neighborhoods may be used.

Algorithm 3 presents the pseudo-code of the BVNS algorithm.

Algorithm 3 BVNS(P_b, $k_{\max}$).

```
1:  k ← 1
2:  repeat
3:      P' ← Shake(P_b, k)
4:      P'' ← LocalSearch(P')
5:      if f(P'') < f(P_b) then
6:          k ← 1
7:          P_b ← P''
8:      else
9:          k ← k + 1
10:     end if
11: until k = k_max
12: return P_b
```

The method requires two inputs: the initial solution, P_b; and the maximum neighborhood to be explored during the search, $k_{\max}$. The initial solution of BVNS is generated by the constructive procedure described in Section 2.1. This constructive procedure generates 100 solutions, and the best solution is then used as the initial solution for the BVNS algorithm.

The BVNS algorithm starts from the first neighborhood considered (step 1), and the following steps are executed until the maximum predefined neighborhood, $k_{\max}$, is reached (steps 2–12). At each iteration, the incumbent solution P_b is perturbed with the *Shake* procedure (step 3). This method randomly selects a solution in the neighborhood under exploration at the current iteration. The shake method is designed to increase the diversity of the search and therefore the centers involved in the exchange are selected at random at each iteration. Once the perturbed solution P' has been obtained, the local search method described in Section 2.1 is then applied, and a local optimum P'' is found in $N_1(P')$ (step 4).

Finally, the neighborhood change stage (steps 5–10) is executed in order to select the subsequent neighborhood to be explored. In particular, if the improved solution P'' is better than the incumbent solution P_b in terms of the objective function value, then P_b is updated and the search starts again from the first neighborhood (steps 6 and 7). Otherwise, the search continues with the next neighborhood (step 9). The method stops when no improvement is found in any neighborhood considered, and the best solution found during the search is returned (step 12).

To close this section, it should be mentioned that the previous algorithms, GRASP and BVNS, have also been hybridized in order to analyze the viability of this combination by taking advantages of their strengths. The hybridization consists of generating solutions using the GRASP constructive procedure, and of replacing the local search method with a complete BVNS algorithm. Specifically, the GRASP constructive procedure is executed 100 times, that is, 100 initial solutions are built and all of them are subsequently improved including them in a single iteration of a BVNS algorithm. The following sections present the computational results for three algorithms.

3. Computational results

This section presents and discusses the results of the computational testing conducted with the three algorithms proposed in this paper over the same set of instances considered in Albareda-Sambola et al. (2015). These instances are available at OR-library[1], see Beasley (1990). A total of 132 instances were solved on an Intel Core 2 Duo E8400 (3 GHz) with 4 GB RAM, and the algorithms were implemented using Java 8.

Albareda-Sambola et al. (2015) divided the instances into three sets: a set of small instances and two sets of larger instances. The first set of instances, made up of 40 small-sized problems, and the second set of instances, 68 large-sized problems, were generated using the instances named pmed1-pmed4 from the OR-library by selecting the first n points of each instance and by varying the p-values following the indications in Albareda-Sambola et al. (2015). Similarly, the instances of the third set, 24 large-sized problems, were generated in the same way as the previous instances except that pmed6, pmed7 and pmed8 were used from the OR-library. See Table 1, where all the combinations of n and p are shown and the number of possible feasible solutions are calculated. Note that the small-sized problems consider $10 \leq n \leq 50$ and the range of the large-sized problems is $60 \leq n \leq 200$ as in Albareda-Sambola et al. (2015).

The results are divided into two parts: preliminary and final experiments. The former are devoted to the selection of the best construction function and the best parameter setting for the GRASP and the BVNS algorithms, while the latter compare the performance of the final results obtained by the three final proposed algorithms.

3.1. Preliminary experiments

In order to select the best combination of parameters for the GRASP and the BVNS algorithms without causing any overfitting, a representative subset of a total of 33 instances (25% of the total set of instances) was randomly selected. Each problem was solved 50 times to show the robustness of the proposed algorithms. Moreover, to facilitate the comparison between the algorithms, the same performance metrics are reported in Tables 2 and 3: the average objective function value (columns named Average OF); the standard deviation (Std. Dev.); the average execution time of the algo-

[1] http://people.brunel.ac.uk/~mastjjb/jeb/info.html

Table 1
Number of feasible solutions.

	$p = 5$	$p = 10$	$p = 20$	$p = 30$	$p = 50$	$p = 80$
$n = 10$	252					
$n = 20$	15504	184756				
$n = 30$	142506	30045015				
$n = 40$	658008	$8.48 \cdot 10^{8}$	$1.38 \cdot 10^{11}$			
$n = 50$		$1.03 \cdot 10^{10}$	$4.71 \cdot 10^{13}$			
$n = 60$		$7.54 \cdot 10^{10}$	$4.19 \cdot 10^{15}$	$1.18 \cdot 10^{17}$		
$n = 70$		$3.97 \cdot 10^{11}$	$1.62 \cdot 10^{17}$	$5.53 \cdot 10^{19}$		
$n = 80$		$1.65 \cdot 10^{12}$	$3.54 \cdot 10^{18}$	$8.87 \cdot 10^{21}$		
$n = 90$		$5.72 \cdot 10^{12}$	$5.10 \cdot 10^{19}$	$6.73 \cdot 10^{23}$	$5.99 \cdot 10^{25}$	
$n = 100$		$1.73 \cdot 10^{13}$	$5.36 \cdot 10^{20}$	$2.94 \cdot 10^{25}$	$1.01 \cdot 10^{29}$	
$n = 150$			$3.63 \cdot 10^{24}$	$3.22 \cdot 10^{31}$	$2.01 \cdot 10^{40}$	$6.66 \cdot 10^{43}$
$n = 200$			$1.61 \cdot 10^{27}$	$4.10 \cdot 10^{35}$	$4.54 \cdot 10^{47}$	$1.65 \cdot 10^{57}$

Table 2
GRASP methodology.

Method	α	Average OF	Std. Dev.	CPU Time	#Best
C1	Random	103.29	5.43	13.52	9
	0.00	105.27	8.84	11.14	6
	0.25	103.53	7.71	18.20	8
	0.50	103.26	6.64	16.35	11
	0.75	103.11	6.07	17.55	23
C2	*Random*	**102.88**	7.24	14.68	**23**
	0.00	123.64	11.64	13.77	3
	0.25	103.85	6.89	14.45	11
	0.50	103.15	6.40	15.71	14
	0.75	103.16	6.56	16.86	15
Random	- - -	103.41	6.34	8.96	9

rithm measured in seconds (CPU Time); and, finally, the number of times that the algorithm is able to attain the best value (#Best).

Table 2 shows the computational results for the GRASP algorithm, while using each of the two strategies to construct solutions: either random (C1) or greedy (C2) selection of the initial centers. The values for the α parameter considered are $\alpha = \{0, 0.25, 0.50, 0.75\}$. Furthermore, in order to accelerate the algorithm, instead of considering $\alpha = 1$, the solutions have been randomly built since it is unnecessary to evaluate all candidates in order to calculate the RCL. All the variants require similar computing time except, naturally, the random construction, and hence we can focus on the quality of the solutions for the selection of the best configuration. The best results are obtained when considering a different random value for α in the C2 variant in terms of the number of the best solutions found and of the average OF.

Similarly, Table 3 presents the results for the preliminary experimentation designed to obtain the best value for the parameters α and β in the BVNS algorithm. Table 3 includes the results for various values of $k_{\max}$. In our case, four different values are chosen in the shake phase, $k_{max} = \lceil p \cdot \beta \rceil$ with $\beta = \{0.1, 0.25, 0.50, 0.75\}$. On analyzing the β parameter, it appears that the larger the value, the better the results (the best results are obtained with $\beta = 0.75$), but this is also the most time-consuming one. The rationale behind this behavior is that in the *p*NCP, it is more important to diversify than to intensify the search, thereby allowing the algorithm to escape from the locally optimal value. However, regarding the results when considering different α values, no significant differences can be found between the two methods in the construction of solutions. These results are in line with those of Hansen et al. (2017),

Table 3
BVNS methodology.

		C1				C2			
α	β	Average OF	Std. Dev.	CPU Time	#Best	Average OF	Std. Dev.	CPU Time	#Best
Random	0.10	102.95	9.00	13.58	20	102.64	8.98	18.48	22
	0.25	102.45	7.90	16.98	24	102.22	8.59	22.66	23
	0.50	102.28	8.48	21.59	26	101.98	7.63	29.17	23
	0.75	102.14	7.53	31.51	25	**101.52**	8.65	33.17	**30**
0.00	0.10	104.51	9.72	13.83	23	111.12	116.10	17.69	14
	0.25	103.63	11.89	19.57	21	106.50	30.58	23.17	22
	0.50	103.11	10.72	25.98	25	104.59	15.96	27.95	23
	0.75	102.67	9.63	32.64	24	104.00	13.25	33.58	25
0.25	0.10	103.07	8.86	15.55	23	103.54	8.44	15.71	21
	0.25	102.60	7.17	19.18	22	102.98	7.76	21.29	23
	0.50	102.23	8.02	24.50	23	102.34	7.94	32.87	24
	0.75	102.03	6.47	37.48	27	102.50	9.25	39.11	25
0.50	0.10	102.85	6.61	14.71	19	102.45	9.61	20.14	24
	0.25	102.47	7.80	18.12	21	102.31	6.54	22.17	21
	0.50	101.98	5.74	23.01	27	102.18	9.67	28.44	26
	0.75	101.73	5.81	30.25	24	102.11	7.48	34.95	23
0.75	0.10	102.65	7.68	15.64	23	102.37	6.35	21.19	22
	0.25	101.73	7.54	17.19	23	101.93	7.21	24.68	25
	0.50	101.90	5.38	23.34	27	101.95	7.67	29.46	26
	0.75	101.83	5.06	31.86	25	101.56	5.11	36.41	28
				Random					
α	k_{max}	Average OF	Std. Dev.	CPU Time	#Best				
- - -	0.10	102.91	7.73	12.87	19				
	0.25	102.23	8.22	14.76	24				
	0.50	102.29	7.05	19.13	22				
	0.75	102.03	6.54	22.08	26				

Table 4
Comparison among GRASP, BVNS, Hybrid algorithms and the optimal solutions in the small-sized instances.

				GRASP			BVNS			Hybrid		
Instance	n	p	Best	OF	CPU Time	gap	OF	CPU Time	gap	OF	CPU Time	gap
pmed1	10	5	84	84	0.04	0.00%	84	0.02	0.00%	84	0.02	0.00%
pmed1	20	5	120	120	0.02	0.00%	120	0.02	0.00%	120	0.04	0.00%
pmed1	20	10	95	95	0.15	0.00%	95	0.08	0.00%	95	0.12	0.00%
pmed1	30	5	126	126	0.06	0.00%	126	0.03	0.00%	126	0.09	0.00%
pmed1	30	10	95	95	0.13	0.00%	95	0.13	0.00%	95	0.28	0.00%
pmed1	40	5	144	149	0.07	3.47%	149	0.06	3.47%	144	0.24	0.00%
pmed1	40	10	111	113	0.28	1.80%	111	0.21	0.00%	111	0.57	0.00%
pmed1	40	20	89	89	0.66	0.00%	89	0.52	0.00%	89	1.67	0.00%
pmed1	50	10	110	111	0.35	0.91%	111	0.32	0.91%	111	0.94	0.91%
pmed1	50	20	89	91	0.89	2.25%	91	0.94	2.25%	89	3.22	0.00%
pmed2	10	5	121	128	0.00	5.79%	128	0.01	5.79%	128	0.01	5.79%
pmed2	20	5	147	147	0.02	0.00%	147	0.02	0.00%	147	0.05	0.00%
pmed2	20	10	99	99	0.05	0.00%	99	0.09	0.00%	99	0.10	0.00%
pmed2	30	5	169	169	0.03	0.00%	169	0.03	0.00%	169	0.10	0.00%
pmed2	30	10	110	110	0.12	0.00%	110	0.11	0.00%	110	0.36	0.00%
pmed2	40	5	164	164	0.06	0.00%	164	0.06	0.00%	164	0.20	0.00%
pmed2	40	10	112	112	0.20	0.00%	132	0.23	17.86%	112	0.56	0.00%
pmed2	40	20	96	96	0.50	0.00%	96	0.53	0.00%	96	1.68	0.00%
pmed2	50	10	140	140	0.33	0.00%	140	0.33	0.00%	140	0.84	0.00%
pmed2	50	20	99	99	0.82	0.00%	99	0.84	0.00%	99	3.28	0.00%
pmed3	10	5	77	77	0.00	0.00%	77	0.01	0.00%	77	0.01	0.00%
pmed3	20	5	145	145	0.01	0.00%	145	0.06	0.00%	145	0.06	0.00%
pmed3	20	10	77	77	0.06	0.00%	77	0.04	0.00%	77	0.10	0.00%
pmed3	30	5	157	157	0.03	0.00%	157	0.03	0.00%	157	0.10	0.00%
pmed3	30	10	122	122	0.12	0.00%	122	0.15	0.00%	122	0.32	0.00%
pmed3	40	5	157	157	0.07	0.00%	157	0.10	0.00%	157	0.20	0.00%
pmed3	40	10	105	105	0.22	0.00%	105	0.23	0.00%	105	0.56	0.00%
pmed3	40	20	77	77	0.50	0.00%	77	0.67	0.00%	77	1.70	0.00%
pmed3	50	10	125	125	0.00	0.00%	125	0.32	0.00%	125	0.86	0.00%
pmed3	50	20	87	87	0.35	0.00%	87	0.82	0.00%	87	2.77	0.00%
pmed4	10	5	126	126	0.80	0.00%	126	0.01	0.00%	126	0.01	0.00%
pmed4	20	5	139	139	0.02	0.00%	139	0.02	0.00%	139	0.04	0.00%
pmed4	20	10	125	125	0.05	0.00%	125	0.05	0.00%	125	0.11	0.00%
pmed4	30	5	173	173	0.03	0.00%	173	0.03	0.00%	173	0.09	0.00%
pmed4	30	10	122	122	0.10	0.00%	122	0.11	0.00%	122	0.29	0.00%
pmed4	40	5	175	175	0.05	0.00%	175	0.06	0.00%	175	0.19	0.00%
pmed4	40	10	122	122	0.21	0.00%	122	0.20	0.00%	122	0.54	0.00%
pmed4	40	20	85	85	0.47	0.00%	85	0.52	0.00%	85	1.73	0.00%
pmed4	50	10	126	126	0.34	0.00%	126	0.29	0.00%	126	0.93	0.00%
pmed4	50	20	91	92	0.90	1.10%	92	0.83	1.10%	91	3.41	0.00%
				118.78	0.23	0.38%	119.23	0.23	0.78%	118.53	0.71	0.17%

who state that the initial solution generation is not a critical phase in BVNS. Notice that an increase in the maximum neighborhood size will result in larger computational times. However, it does not always lead to better solutions. Therefore, it is helpful to perform this analysis of the β parameter to fix k_{max}.

Summarizing, the comparison between the GRASP, the BVNS algorithms and their hybridization is performed in the following section, whereby the best parameters obtained in the preliminary experiments above are employed. Note that, in several cases, there are no significant differences between certain parameters, as happens in the BVNS algorithm where it is of no consequence which strategy and α parameter are used to construct solutions because the quality of the solutions only depends on the perturbation parameter. Hence, the GRASP algorithm uses the second strategy to construct solutions (C2) and uses a random α parameter. On the other hand, the BVNS algorithm fixes the perturbation parameter with $\beta = 0.75$ and uses the same construction.

3.2. Comparison between GRASP and BVNS versus their hybrid

Once the best parameters for the GRASP and the BVNS algorithms have been chosen, their comparison is performed as previously mentioned. In order to include a more thorough analysis of the algorithms, a hybrid algorithm combining the GRASP and the BVNS has been incorporated using the same parameters. To this end, all the instances are included in the results.

Tables 4–7 show the results obtained for the 132 instances where 100 solutions were generated for each instance. Column 1 of these tables includes the name for the instances, and columns 2 and 3 contain the combination of n and p. Column 4 shows the optimal values obtained by Albareda-Sambola et al. (2015) for the small-sized instances in Table 4 and the best-known values obtained by our algorithms for the large-sized instances in Tables 5–7. Columns 5, 6 and 7 show the results obtained using the GRASP algorithm, columns 8, 9 and 10 show the results obtained by the BVNS algorithm, and columns 11, 12 and 13 show the results obtained by the hybrid GRASP-BVNS algorithm. Specifically, columns 5, 8, and 11 report the value of the solution found by the GRASP, the BVNS and the hybrid GRASP-BVNS algorithms, respectively. Similarly, columns 6, 9, and 12 present the CPU time used, and columns 7, 10, and 13 provide the gap calculated as the difference between the solution obtained and the best solution found and this difference is divided by the best solution found (as a percentage).

At this point, each set of problems are analyzed separately. Regarding the small-sized set of instances (see Table 4), it can be observed that the GRASP and the BVNS algorithms are able to find 34 out of 40 optimal solutions, while the hybrid GRASP-BVNS algorithm is able to find the optimal solutions in 38 out of 40 in-

Table 5
Comparison among GRASP, BVNS, and Hybrid algorithms in the large-sized instances.

				GRASP			BVNS			Hybrid		
Instance	n	p	Best	OF	CPU Time	gap	OF	CPU Time	gap	OF	CPU Time	gap
pmed1	60	10	112	112	0.48	0.00%	115	0.41	2.68%	112	1.26	0.00%
pmed1	60	20	91	91	1.14	0.00%	95	1.18	4.40%	91	5.08	0.00%
pmed1	60	30	89	89	1.94	0.00%	89	2.04	0.00%	89	11.50	0.00%
pmed1	70	10	119	131	0.55	10.08%	131	0.55	10.08%	119	1.84	0.00%
pmed1	70	20	99	99	1.65	0.00%	99	1.65	0.00%	99	7.74	0.00%
pmed1	70	30	73	85	2.92	16.44%	91	3.04	24.66%	73	18.85	0.00%
pmed1	80	10	133	134	0.72	0.75%	133	0.74	0.00%	133	2.44	0.00%
pmed1	80	20	105	112	2.12	6.67%	108	2.32	2.86%	105	11.64	0.00%
pmed1	80	30	91	99	3.82	8.79%	99	4.10	8.79%	91	29.16	0.00%
pmed1	90	10	133	134	0.91	0.75%	133	0.91	0.00%	133	3.34	0.00%
pmed1	90	20	108	110	2.73	1.85%	108	2.86	0.00%	108	15.05	0.00%
pmed1	90	30	91	105	5.03	15.38%	95	5.60	4.40%	91	38.29	0.00%
pmed1	90	50	70	81	10.87	15.71%	71	12.21	1.43%	70	103.71	0.00%
pmed1	100	10	133	137	1.43	3.01%	134	1.34	0.75%	133	4.36	0.00%
pmed1	100	20	108	116	3.63	7.41%	112	3.80	3.70%	108	19.32	0.00%
pmed1	100	30	97	104	6.84	7.22%	103	7.11	6.19%	97	51.51	0.00%
pmed1	100	50	74	82	14.61	10.81%	77	16.37	4.05%	74	143.32	0.00%
pmed2	60	10	140	140	0.41	0.00%	140	0.41	0.00%	140	1.43	0.00%
pmed2	60	20	99	102	1.13	3.03%	102	1.17	3.03%	99	6.10	0.00%
pmed2	60	30	96	96	1.90	0.00%	96	2.04	0.00%	96	13.61	0.00%
pmed2	70	10	138	138	0.53	0.00%	140	0.53	1.45%	138	1.97	0.00%
pmed2	70	20	102	108	1.62	5.88%	102	1.68	0.00%	102	8.83	0.00%
pmed2	70	30	96	97	2.83	1.04%	96	2.99	0.00%	96	20.65	0.00%
pmed2	80	10	138	138	0.68	0.00%	145	0.73	5.07%	138	2.78	0.00%
pmed2	80	20	109	118	2.11	8.26%	118	2.23	8.26%	109	12.40	0.00%
pmed2	80	30	97	100	3.80	3.09%	100	4.25	3.09%	97	30.84	0.00%
pmed2	90	10	140	143	0.88	2.14%	145	0.91	3.57%	140	3.56	0.00%
pmed2	90	20	109	120	2.73	10.09%	118	2.89	8.26%	109	16.52	0.00%
pmed2	90	30	97	102	5.12	5.15%	102	5.51	5.15%	97	41.52	0.00%
pmed2	90	50	96	96	10.99	0.00%	96	12.19	0.00%	96	104.57	0.00%
pmed2	100	10	135	138	1.12	2.22%	135	1.11	0.00%	135	3.97	0.00%
pmed2	100	20	109	118	3.43	8.26%	112	3.67	2.75%	109	18.81	0.00%
pmed2	100	30	96	102	6.49	6.25%	102	7.03	6.25%	96	50.26	0.00%
pmed2	100	50	96	96	14.42	0.00%	96	15.97	0.00%	96	145.13	0.00%
				110.97	3.58	4.71%	109.94	3.87	3.56%	106.44	27.98	0.00%

stances. Furthermore, the average gap is 0.38% for the GRASP algorithm, 0.78% for the BVNS algorithm, and 0.17% for the hybrid GRASP-BVNS algorithm. However, the average CPU time spent by the hybrid GRASP-BVNS algorithm (0.71 s) is much larger than that consumed by the GRASP (0.23 s) or the BVNS algorithm (0.23 s). Therefore, we can conclude that for the small-sized instances the hybrid algorithm is more than 10 times slower than the GRASP algorithm and the BVNS algorithm but it is able to find more optimal solutions.

On observing the large-sized set of instances (see Tables 5–7), similar conclusions can now be highlighted. The hybrid GRASP-BVNS algorithm is able to find the best solution on all the instances, while in contrast the GRASP algorithm is able to find the best solutions in just 25 out of 92 and the BVNS algorithm in 29 out of 92 instances. Here, in the large-sized set of instances, the optimal solutions remained unavailable, and hence the comparison is performed against the best solutions obtained by the GRASP, the BVNS and the hybrid algorithms. Again, the average gap considering the 92 large-sized instances is 5.40% for the GRASP algorithm, compared with the 4.90% for the BVNS algorithm and 0% for the hybrid version. Furthermore, the average CPU time consumed by the GRASP algorithm is 11.53 s, 12.38 s by the BVNS algorithm and 124.60 s by the hybrid GRASP-BVNS algorithm.

On analyzing these results, we can conclude that the hybrid algorithm between the GRASP and the BVNS algorithms performs much better than do the GRASP and the BVNS separately. Nevertheless, the hybridization of these two algorithms is much more time-consuming than the individual GRASP and BVNS algorithms. Therefore, the decision-maker must assess whether interest is focused on the quality of the solution or on the time consumed.

4. Conclusions and future research

This paper solves an interesting problem, known as the *p*-next center problem, which models real situations in humanitarian logistics. For this problem, two metaheuristics are proposed, GRASP and BVNS, which are later combined into a hybrid algorithm. A wide set of instances with different sizes is solved and the algorithms are able to obtain optimal or near-optimal solutions in short computing times. For the small-sized instances (up to 50 nodes), the GRASP and the BVNS algorithms find the optimal value in the majority of the instances (in all except six instances), on average in less than one second. However, the hybrid algorithm obtains the optimal value in nearly all the instances (all except two instances), on average in less than one second as well. Furthermore, for the two sets of large-sized instances (up to 200 nodes), the hybrid GRASP-BVNS algorithm obtains better solutions than the GRASP algorithm or the BVNS algorithm separately. However, the hybrid GRASP-BVNS algorithm consumes more CPU time than do either of the GRASP and the BNVS algorithms, which obtain very good solutions within a reasonable computational effort.

As future work, it would be interesting to solve an extension of the *p*-next center problem, in which all users are assumed to be assigned to their reference center. In the case where this center is unavailable, then the users are redirected to the next closest center to that unavailable reference center (backup center), and in the unlikely case that this center is also unavailable, then the users are redirected to the next closest center to the backup center, and so on. Of course, in the worst-case scenario, all the centers would be unavailable except one.

Table 6
Comparison among the GRASP, BVNS, and Hybrid algorithms in the large-sized instances.

				GRASP			BVNS			Hybrid		
Instance	n	p	Best	OF	CPU Time	gap	OF	CPU Time	gap	OF	CPU Time	gap
pmed3	60	10	124	124	0.42	0.00%	125	0.43	0.81%	124	1.47	0.00%
pmed3	60	20	97	97	1.14	0.00%	97	1.18	0.00%	97	5.14	0.00%
pmed3	60	30	73	79	1.91	8.22%	73	2.08	0.00%	73	10.83	0.00%
pmed3	70	10	121	127	0.57	4.96%	127	0.55	4.96%	121	2.15	0.00%
pmed3	70	20	97	98	1.63	1.03%	105	1.65	8.25%	97	8.66	0.00%
pmed3	70	30	82	87	2.86	6.10%	87	3.00	6.10%	82	19.58	0.00%
pmed3	80	10	121	127	0.71	4.96%	124	0.72	2.48%	121	2.89	0.00%
pmed3	80	20	93	106	2.17	13.98%	97	2.24	4.30%	93	12.04	0.00%
pmed3	80	30	86	93	3.79	8.14%	87	4.23	1.16%	86	29.93	0.00%
pmed3	90	10	148	148	0.87	0.00%	148	0.91	0.00%	148	2.82	0.00%
pmed3	90	20	105	111	2.98	5.71%	112	2.91	6.67%	105	12.60	0.00%
pmed3	90	30	93	97	5.02	4.30%	97	5.39	4.30%	93	33.52	0.00%
pmed3	90	50	93	93	10.92	0.00%	93	12.35	0.00%	93	89.92	0.00%
pmed3	100	10	151	152	1.13	0.66%	151	1.10	0.00%	151	3.56	0.00%
pmed3	100	20	113	119	3.35	5.31%	119	3.50	5.31%	113	17.13	0.00%
pmed3	100	30	93	102	6.41	9.68%	97	6.95	4.30%	93	44.94	0.00%
pmed3	100	50	93	93	14.64	0.00%	93	16.95	0.00%	93	124.77	0.00%
pmed4	60	10	135	135	0.41	0.00%	135	0.41	0.00%	135	1.46	0.00%
pmed4	60	20	93	96	1.13	3.23%	93	1.18	0.00%	93	5.43	0.00%
pmed4	60	30	79	84	1.94	6.33%	83	2.04	5.06%	79	13.89	0.00%
pmed4	70	10	146	146	0.58	0.00%	146	0.56	0.00%	146	2.04	0.00%
pmed4	70	20	102	102	1.61	0.00%	111	1.66	8.82%	102	8.43	0.00%
pmed4	70	30	85	85	2.80	0.00%	94	2.99	10.59%	85	19.17	0.00%
pmed4	80	10	146	146	0.70	0.00%	147	0.72	0.68%	146	2.81	0.00%
pmed4	80	20	114	124	2.13	8.77%	119	2.28	4.39%	114	12.05	0.00%
pmed4	80	30	91	101	3.91	10.99%	94	4.17	3.30%	91	29.39	0.00%
pmed4	90	10	147	154	0.91	4.76%	149	0.94	1.36%	147	3.63	0.00%
pmed4	90	20	112	120	2.77	7.14%	123	2.91	9.82%	112	16.47	0.00%
pmed4	90	30	92	98	5.06	6.52%	96	5.43	4.35%	92	43.52	0.00%
pmed4	90	50	82	83	10.98	1.22%	83	12.50	1.22%	82	102.93	0.00%
pmed4	100	10	147	148	1.12	0.68%	147	1.28	0.00%	147	4.71	0.00%
pmed4	100	20	119	120	3.51	0.84%	124	4.10	4.20%	119	21.90	0.00%
pmed4	100	30	96	112	6.51	16.67%	106	7.04	10.42%	96	57.82	0.00%
pmed4	100	50	82	86	14.67	4.88%	84	16.19	2.44%	82	154.18	0.00%
				111.56	3.56	4.27%	110.76	3.90	3.39%	107.38	27.11	0.00%

Table 7
Comparison among GRASP, BVNS, and Hybrid algorithms in the large-sized instances.

				GRASP			BVNS			Hybrid		
Instance	n	p	Best	OF	CPU Time	gap	OF	CPU Time	gap	OF	CPU Time	gap
pmed6	150	20	79	81	5.21	2.53%	85	4.80	7.59%	79	33.86	0.00%
pmed6	150	30	71	82	8.98	15.49%	80	9.67	12.68%	71	77.19	0.00%
pmed6	150	50	62	68	20.26	9.68%	62	22.35	0.00%	62	200.17	0.00%
pmed6	150	80	56	56	40.81	0.00%	56	47.16	0.00%	56	479.01	0.00%
pmed6	200	20	79	83	8.19	5.06%	81	8.83	2.53%	79	49.78	0.00%
pmed6	200	30	72	81	16.64	12.50%	80	17.93	11.11%	72	150.29	0.00%
pmed6	200	50	68	74	39.64	8.82%	71	43.79	4.41%	68	493.55	0.00%
pmed6	200	80	54	56	83.99	3.70%	55	95.01	1.85%	54	1151.12	0.00%
pmed7	150	20	69	70	4.38	1.45%	69	4.26	0.00%	69	22.90	0.00%
pmed7	150	30	62	71	8.52	14.52%	69	8.43	11.29%	62	66.10	0.00%
pmed7	150	50	59	60	20.06	1.69%	59	20.52	0.00%	59	206.33	0.00%
pmed7	150	80	59	59	41.34	0.00%	59	42.66	0.00%	59	415.03	0.00%
pmed7	200	20	73	84	8.25	15.07%	84	7.62	15.07%	73	44.05	0.00%
pmed7	200	30	68	79	15.83	16.18%	72	15.87	5.88%	68	129.25	0.00%
pmed7	200	50	63	69	38.23	9.52%	66	39.01	4.76%	63	505.03	0.00%
pmed7	200	80	52	59	81.61	13.46%	58	89.28	11.54%	52	1213.66	0.00%
pmed8	150	20	74	83	4.29	12.16%	81	4.21	9.46%	74	23.76	0.00%
pmed8	150	30	61	73	8.39	19.67%	70	8.38	14.75%	61	64.84	0.00%
pmed8	150	50	58	58	20.07	0.00%	60	20.41	3.45%	58	194.59	0.00%
pmed8	150	80	58	58	40.75	0.00%	58	41.51	0.00%	58	430.29	0.00%
pmed8	200	20	84	92	7.82	9.52%	89	7.67	5.95%	84	41.21	0.00%
pmed8	200	30	77	88	15.91	14.29%	86	15.29	11.69%	77	122.14	0.00%
pmed8	200	50	68	72	38.20	5.88%	72	41.74	5.88%	68	394.95	0.00%
pmed8	200	80	68	68	81.95	0.00%	68	88.43	0.00%	68	1140.05	0.00%
				71.83	27.47	7.97%	70.42	29.37	5.83%	66.42	318.71	0.00%

Additionally, as Albareda-Sambola et al. (2015) point out, it would be of major interest to consider capacity constraints in the centers, since in the real situations associated with this model, hospitals, refugee camps, and rescue centers can experience this kind of limitation.

Furthermore, another problem similar to the *p*NCP that is very realistic appears when the users know beforehand that their closest center has become unavailable. In that case, the users must go directly to their second-closest center. This problem is known as the 2-neighbor *p*-center problem, whose objective is to minimize the distance to the second-closest center. The generalization of this problem is the α-neighbor *p*-center problem whose objective is to minimize the distance to the α-closest center. Yet another similar problem could be considered, called the *p*-second center problem (pSCP), which consists of locating p out of n centers in order to minimize the maximum distance between users and their closest center and also the distance from these users to their second-closest center. Here, users are not certain until the last moment whether they will go to the closest center or the second-closest center as in the 2-neighbor *p*-center problem. Therefore, the *p*SCP can be formally defined as follows:

$$\min_{\substack{P\subset V\\ |P|=p}} \max_{i\in V} \left\{ \min_{j\in P} d_{ij} + \min_{\substack{k\in P\\ k\neq j}} d_{ik} \right\}$$

Acknowledgments

J. Sánchez-Oro is supported by the Spanish Ministry of "Economía y Competitividad", Grant Refs. TIN2015-65460-C2-2-P and TIN2014-54806-R.

A.D. López-Sánchez and A.G. Hernández-Díaz acknowledge support from the Spanish Ministry of Science and Innovation through Projects ECO2013-47129-C4-1-R and ECO2016-76567-C4-1-R.

The authors wish to express their gratitude to M. Albareda-Sambola, Y. Hinojosa, A. Marín and J. Puerto for their collaboration in the provision of all the instances and previous results.

References

Albareda-Sambola, M., Hinojosa, Y., Marín, A., Puerto, J., 2015. When centers can fail: a close second opportunity. Comput. Operat. Res. 62 (Supplement C), 145–156.

Beasley, J.E., 1990. OR-Library: distributing test problems by electronic mail. J. Operat. Res. Soc. 41 (11), 1069–1072.

Calik, H., Tansel, B.C., 2013. Double bound method for solving the *p*-center location problem. Comput. Operat. Res. 40 (12), 2991–2999.

Daskin, M.S., 1995. Network and Discrete Location. John Wiley & Sons, Inc..

Davidovic, T., Ramljak, D., Selmic, M., Teodorovic, D., 2011. Bee colony optimization for the *p*-center problem. Comput. Operat. Res. 38 (10), 1367–1376.

Duarte, A., Pantrigo, J., Pardo, E., Sánchez-Oro, J., 2016. Parallel variable neighbourhood search strategies for the cutwidth minimization problem. IMA J. Manag. Math. 27 (1), 55.

Elloumi, S., Labbé, M., Pochet, Y., 2004. A new formulation and resolution method for the *p*-center problem. INFORMS J. Comput. 16 (1), 84–94.

Feo, T.A., Resende, M.G.C., 1989. A probabilistic heuristic for a computationally difficult set covering problem. Operat. Res. Lett. 8 (2), 67–71.

Feo, T.A., Resende, M.G.C., 1995. Greedy randomized adaptive search procedures. J. Glob. Optim. 6 (2), 109–133.

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

Festa, P., Resende, M.G.C., 2008. An annotated bibliography of GRASPâpart i: algorithms. Int. Trans. Operat. Research 16 (1), 1–24.

Festa, P., Resende, M.G.C., 2009. An annotated bibliography of GRASPâpart II: applications. Int. Trans. Operat. Res. 16 (2), 131–172.

Hakimi, S.L., 1964. Optimum locations of switching centers and the absolute centers and medians of a graph. Operat. Res. 12 (3), 450–459.

Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S., 2017. Variable neighborhood search: basics and variants. EURO J. Comput. Optim. 5 (3), 423–454.

Irawan, C.A., Salhi, S., Drezner, Z., 2016. Hybrid meta-heuristics with VNS and exact methods: application to large unconditional and conditional vertex *p*-centre problems. J. Heurist. 22 (4), 507–537.

Martínez-Merino, L.I., Albareda-Sambola, M., RodrÃguez-Chía, A.M., 2017. The probabilistic *p*-center problem: planning service for potential customers. Eur. J. Operat. Res. 262 (2), 509–520.

Minieka, E., 1970. The m-center problem. SIAM Rev. 12 (1), 138–139.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. Operat. Res. 24 (11), 1097–1100.

Mladenović, N., Labbé, M., Hansen, P., 2003. Solving the *p*-center problem with tabu search and variable neighborhood search. Networks 42 (1), 48–64.

Quintana, J., Sánchez-Oro, J., Duarte, A., 2016. Efficient greedy randomized adaptive search procedure for the generalized regenerator location problem. Int. J. Comput. Intell. Syst. 9 (6), 1016–1027.

Resende, M.G., Ribeiro, C.C., 2010. Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications. Springer US, Boston, MA, pp. 283–319.

Sánchez-Oro, J., Sevaux, M., Rossi, A., Martí, R., Duarte, A., 2015. Solving dynamic memory allocation problems in embedded systems with parallel variable neighborhood search strategies. Electron. Notes Discrete Math. 47, 85–92.

Yin, A.-H., Zhou, T., Ding, J., Zhao, Q.-J., Lv, Z.-P., 2017. Greedy randomized adaptive search procedure with path-relinking for the vertex *p*-center problem. J. Comput. Sci. Technol. 32 (6), 1319–1334.