International Journal for Traffic and Transport Engineering, 2021, 11(2): 294 - 309

DOI: http://dx.doi.org/10.7708/ijtte2021.11(2).09 UDC: 656.02:004.421

FILTERED VARIABLE NEIGHBORHOOD SEARCH METHOD FOR THE P-NEXT CENTER PROBLEM

Dalibor Ristić[1], Nenad Mladenović[2], Raca Todosijević[3], Dragan Urosević[1, 4]

[1] *School of Computing, Union University, Belgrade, Serbia*

[2] *Khalifa University, Abu Dhabi, United Arab Emirates*

[3] *Polytechnic University of Hauts-de-France, Valenciennes, France*

[4] *Mathematical Institute, University of Belgrade, Belgrade, Serbia*

Received 30 November 2020; accepted 20 January 2021

Abstract: The p-center problem has been the subject of interest in the operational research for a long time. It has been well-known since the middle of the previous century. During the last decade, an extension of the problem, known as the p-next center problem, has been defined in order to handle unexpected incidents that can disable the centers. There are only a few papers and algorithms that address the aforementioned problem and therefore we introduce a new algorithm for solving the p-next center problem based on the Variable Neighborhood Search Method. The proposed algorithm was tested on a set of test instances already known in the literature, and the results show that it returns an optimal or at least near-optimal solution to the problem in a reasonable amount of time. Compared to existing algorithms, it has been shown that the proposed algorithm finds the best known or better solutions.

Keywords: variable neighborhood search, heuristic algorithms, p-next center problem, combinatorial optimization.

1. Introduction

The p-center problem was introduced in 1965 (Hakimi, 1965) and has been the subject of many research projects ever since. The p-center is a discrete optimization problem that represents the identification of *p* functional centers and their assignment to appropriate users, all in order to minimize the maximum weight determined by the pairs (center, user). For example, the problem can be presented as a model of determining the locations of *p* ambulances (centers) that will serve *n* settlements (users) in order for the distance of the farthest village from the assigned ambulance to be minimal, or it can present the problem of minimizing the maximum distance of all the settlements to the nearest of *p* fire stations, refugee reception centers or cultural centers.

Although the p-center has been shown to be NP-hard (Kariv and Hakimi, 1979), over the time, many exact mathematical models and heuristic algorithms have been published that address this problem, such as:

- Exact methods:
    - Covering method by Minieka (1970);
    - Linear programming (LP) model by Ilhan and Pınar (2001);
    - Integer programming (IP) model by

[1] Corresponding author: dalibor.ristic@outlook.com

Elloumi *et al.* (2004);
- Mixed integer programming (MIP) formulation and a set covering based algorithm by Daskin (2013);
- IP models and an exact algorithm based on the decomposition of the models by Calik and Tansel (2013).

- Heuristic algorithms:
  - Two-approximation O(|E|log|E|) algorithm for the unweighted discrete problem, with edges satisfying triangle inequality, by Hochbaum and Shmoys (1985);
  - VNS and Tabu Search algorithms by Mladenovic *et al.* (2003);
  - Genetic algorithm by Pullan (2008);
  - Bee colony optimization for the p-center problem by Davidovic *et al.* (2011).

Despite the fact that it can be said that the p-center problem has been successfully solved, especially in conditions of humanitarian catastrophes or severe weather disasters, it turned out that there is practically a problem of limited capacity or collapse of centers, which further leads to their inability to serve all intended users. The question was what to do in a case of failure of the assigned center. Other problems have been defined, partly in response to this question, such as capacitated p-center, where centers have limited capacity, or conditional p-center, fault tolerant p-center problem, etc. The conditional p-center problem implies that *q* centers already exist and that the set of centers can be expanded further with additional p centers as needed so that the maximum distance between users and assigned centers is minimized, taking into account all q + p centers. Fault tolerant presents the generalization of a p-center problem, where each user is assigned with multiple centers.

Albareda-Sambola *et al.* (2015) presented a logistical solution to the problem of possible catastrophes, called the p-next center problem. The p-next center problem is a variant of the fault tolerant problem and it is derived from the p-center problem, where the possibility of disabling the center is solved by introducing exactly one replacement center. The centers assigned to the user were given names, references and backup center, and the p-next center problem was defined as the problem of minimizing the maximum distance, among all users, to the nearest (reference) center plus the distance to its nearest (backup) center. The reference center is the closest center to the user and in the case it is disabled, the customer service is then transferred to the assigned backup center. Unlike the p-center problem, the solution to the p-next center problem is to minimize the distance, not to the nearest, but to the backup center, where the reference center is visited first. In case there are several centers at a minimum distance from the user, the reference center is the one that has the nearest backup center.

In order to define the problem, the following notation is introduced: let G = (V, E) be an undirected weighted graph where the weights of the branches are determined by the distance between their ends, V is the set of all nodes, and E is the set of branches of the graph. Centers and other users represent the nodes of the graph, and *d(i, j)* is the shortest distance between nodes *i* and *j*, calculated as a result of an algorithm for determining the shortest paths in graph G. The solution to the p-next center problem is a set of centers $P \subset V$, of cardinality *p*, so that the maximum distance among all the users $i \in V$ to the closest center $j \in P$, plus the distance to its closest center $k \in P$ is minimized.

$$f(P) = \min_{P \subset V, |P|=p} \max_{i \in V} \left\{ \min_{j \in P} d(i,j) + \min_{k \in P, k \neq j' \in \arg\min_{j \in P} d(i,j)} d(j',k) \right\} \tag{1}$$

In the paper by Albareda-Sambola *et al.* (2015), along with the definition of the problem, there is also formal proof that the p-next center problem is NP-hard. In the same paper, several exact mathematical models are presented that solve smaller instances of the problem. In addition to these solutions, there are heuristic algorithms for the p-next center problem presented in the paper by Lopez-Sanchez *et al.* (2018). They introduced Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search (VNS) as well as a hybrid version of these algorithms.

GRASP is based on a metaheuristic initially proposed by Feo and Resende (1989). It is presented as a multi-start framework that consists of a construction and a local search phase. In the construction phase, a solution is gradually generated in such a way, that in each iteration, a new element is added to the current solution. The combination of greediness and randomness approach is achieved by randomly selecting the element to join the solution, from a predefined list of candidates, and the candidate list is generated using a function that selects candidates based on the hyper-parameter α that controls the degree of greediness and randomness combination. If $\alpha = 1$, the candidates are chosen completely randomly, and $\alpha = 0$ determines the completely greedy function that selects the candidates based on the most promising criteria. The second phase of the GRASP algorithm is a local search that tries to identify the local optimum. It implements a simple search algorithm that visits adjacent (N_1) solutions in a random order and if it encounters a solution that improves the current one, it accepts the new one as the current solution. The local search is repeated as long as it is possible to find a solution that is better than the current one. The complete algorithm is executed until a predetermined number of solutions is generated.

VNS implements the basic VNS metaheuristic which is explained in detail in the next section. It consists of a shaking and a local search phase that is implemented in the same way as the local search of the GRASP algorithm.

The hybrid version of the algorithm combines GRASP and VNS so that the local search phase of the GRASP algorithm is replaced by a complete VNS algorithm.

All the algorithms from the work of Lopez-Sanchez *et al.* (2018) were tested over the same test set as the solutions from the work of Albaredo-Sambola *et al.* (2015) and it turned out that the hybrid version returns much better solutions compared to others, but also requires much more CPU time. GRASP and VNS algorithms gave results of approximately the same quality.

This paper introduces a new heuristic VNS algorithm that solves the p-next center problem, comparable to the most successful so far, i.e. hybrid algorithm from the work of Lopez-Sanchez *et al.* (2018). To this end, in the next section, through a more detailed description and pseudo-code, we will discuss the proposed algorithm, and in the third section, we will present the obtained results in a table form, in comparison with the mentioned hybrid algorithm. We will end the paper with a short summary and an announcement of our future work.

2. Algorithm

The algorithm we propose as a new solution to the p-next center problem is built upon the Variable Neighborhood Search Method (in the rest of the text VNS). VNS was first introduced by Mladenovic and Hansen (1997) as a generic framework for building search algorithms that guarantees a systematically structured visit to neighboring solutions. The Variable Neighborhood Method relies on search of near and far neighborhoods of the current solution. Neighborhood $N_1(P)$, of a solution P, defines a complete set of solutions that differ from the set P in one center. For example, let P = {1, 2, 3} be the current solution to the p-next center problem for p = 3 and V = {1, 2, 3, 4,..., n} users. Neighborhood $N_1(P)$ = {{1, 2, 4}, {1, 2, 5},..., {1, 2, n}, {1, 4, 3},..., {n, 2, 3 }} is a set of sets obtained by replacing exactly one center from solution P with a new center outside that solution.

Similarly, $N_k(P)$ neighborhood, where $1 \le k \le |P|$, is a set of solutions obtained by replacing exactly k centers from solution P with new centers that are not included in solution P:

$$N_k(P) = \{P \setminus \{u_1, u_2, \dots, u_k\} \cup \{v_1, v_2, \dots, v_k\} | u_1, u_2, \dots, u_k \in P, v_1, v_2, \dots, v_k \in (V \setminus P), u_i \neq u_j, v_i \neq v_j, 1 \le i < j \le n\} \quad (2)$$

The basic VNS implementation contains two phases that are executed alternately: the local search phase and the change of the current neighborhood. The local search phase implements the current solution search algorithm in order to identify the local optimum. The phase of changing the neighborhood represents a jump from the current solution and the locally optimal value to one of the solutions from the k-neighborhood. The solution is chosen on a random sample principle, in order to eliminate the probability of an infinite loop, to which often leads reliance on deterministic rules. VNS as a generic framework suggests, starting with the solution P, and the first jump into the $N_1(P)$ neighborhood (k = 1), and then to more distant (k = k + 1) neighborhoods. If in the k neighborhood a local search procedure identifies a better solution than the current one P, the new solution becomes current and k is again set to 1. Otherwise, the new solution is discarded and the search continues in the k + 1 neighborhood. The search ends if the current solution is not improved in the k_{max} neighborhood.

It was mentioned that the p-next center problem is addressed by the GRASP, VNS and Hybrid algorithms from the work of Lopez-Sanchez *et al.* (2018). The paper states that GRASP insists on the diversification of the solution, and that the advantage of the VNS algorithm is in the intensification of a single predefined solution. GRASP supports a multi-start approach by initially constructing a set of different solutions that it gradually improves later, while VNS is constantly working on intensifying one predefined solution. It turned out that the Hybrid algorithm, as a combination of the two, inherits the good features of both and as such finds the best solutions. This observation motivated us to try to improve the VNS implementation to achieve a sufficient degree of combination of solution diversification and intensification to be able to identify equally good, and better, solutions compared to the hybrid version of the algorithm.

In our VNS implementation of the algorithm for the p-next center problem, we achieve the maximum diversification of the solution by choosing the largest possible k_{max} value, i.e. $k_{max} = p$. On the other hand, the multi-start approach stands out as a key feature of GRASP that results in the superiority of the algorithm in terms of diversification. To nullify this advantage of the GRASP algorithm, we introduce a new hyper-parameter t_{max} as the execution time limit. We end the VNS algorithm when the allowed execution time expires, and in case the maximum (k_{max}) value of the parameter k is reached before the time limit is exceeded, a new execution is initiated.

It is noticeable that the phase of local search of GRASP and VNS algorithm will not always identify the local optimum. To further improve the efficiency of the algorithm, in contrast to the algorithm from the work of Lopez-Sanchez *et al.* (2018) we do not accept the first better solution we come across during the local search, but we try to identify the optimal solution in order to maximize the intensification of the solution. This would entail more CPU time because it would be necessary to process all p * (n - p) possible solutions from the N_1 neighborhood. Many of these solutions will not improve the objective function value, so efficient filtering of these solutions could significantly speed up the local search phase. The objective function value of the current solution P is determined by the maximum value of the distance from the user to the reference center plus the distance to the backup center, taking into account all the users. Let the user with the maximum distance be the so-called critical user u_c, its reference center critical center c_c, and the mentioned distance critical distance f_c. During a local search, we try to exchange the two centers, that is, to replace the center c_{out} of the current solution with a new c_{in} center by improving the value of the target function. In other words, only those solutions that reduce the critical distance f_c come into consideration. The idea is to first check the value of the critical user function and discard all centers that do not result in a reduction of the distance f_c.

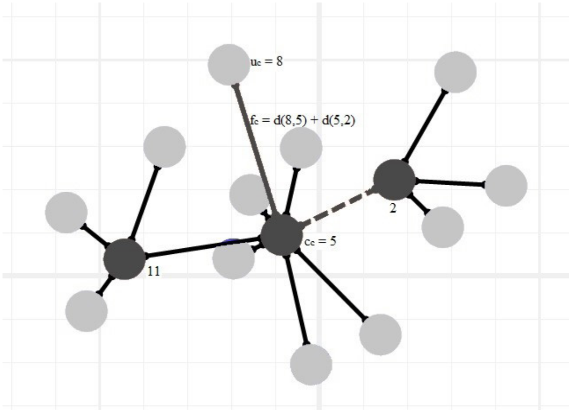

Fig. 1.
Example of Current Solution P = {2, 5, 11} of the p-next Center Problem for n = 15 and p = 3

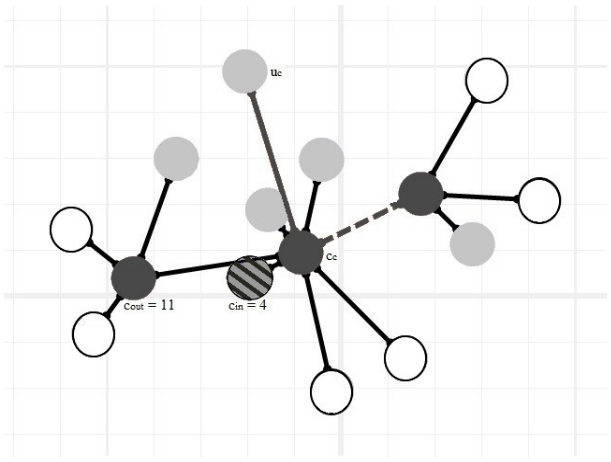

Fig. 2.
Identification of c_{in} and c_{out} Centers to be exchanged during the Local Search phase

For example, let P = {2, 5, 11} be the current solution of the p-next center problem for n = 15 users and p = 3 centers, as shown in Fig. 1. The centers are represented by dark grey and the users by light grey nodes. The critical user is 8, the critical center is 5, and its backup center is 2, so the critical distance f_c is d(8, 5) + d(5, 2). The goal is to determine the optimal pair of centers c_{in} and c_{out}, the exchange of which reduces the current value of the objective function. At first, all potential centers, whose inclusion in the current solution does not relax the distance of the critical user 8 to its (old or new) backup center by passing through the (old or new) reference center, are eliminated. In Fig. 2, users, i.e. potential new centers, which are rejected, are marked in white. Among the remaining users and current centers, a pair (c_{in}, c_{out}) is required whose exchange reduces the f_c value to the minimum. This is the user c_{in} = 4 and the center c_{out} = 11. After their exchange, a new current solution P = {2, 4, 5} was obtained, where the new critical user is u_c = 1, and the critical center is c_c = 2 (Fig. 3). The new critical distance is f_c = d(1, 2) + d(2, 5), where it is guaranteed to be less than the previous d(8, 5) + d(5, 2).

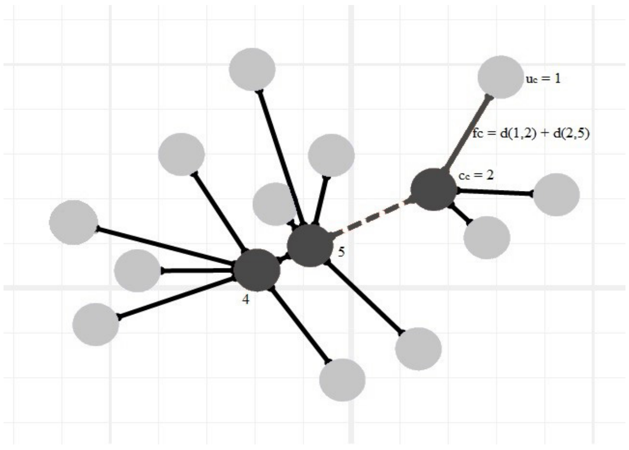

Fig. 3.
New Current Solutions $P = \{2, 4, 5\}$ of the p-next Center Problem for $n = 15$ and $p = 3$ after the Exchange of $c_{in} = 4$ and $c_{out} = 11$ Centers

The previously described and illustrated characteristics of the new VNS implementation are the key to achieving the greatest possible degree of diversification and intensification of the solution. The following is a description of the proposed algorithm and pseudo-code.

To calculate the objective function value (f_{value}), auxiliary functions *f(P, u)* and *f(P, u, c_{in}, c_{out})* are used, which respectively return the value of the user *u* distance to the nearest (reference) center plus the distance to its nearest (backup) center in the current solution P, and referred to the second function, the same just in the case that the c_{out} center of the solution P is replaced by a new c_{in} center ($c_{in} \in V \setminus P$). Also, the function *reference(P, u)* finds the reference center of the user *u* in the solution P. In the case that several centers from P are at the same minimum distance from *u*, the one that has the closest center among other centers of the solution P is chosen as the reference center, that is, any of those who have the next center at the same minimum distance. In the worst case, these functions visit all current centers so that the worst time complexity is O(p).

Algoritam 1: VNS($G = (V, E), P, k_{max}, t_{max}$)

```
1:  $k = 1; f_{value} = \infty$
2:  Do
    Generating a solution at random from kth neighborhood:
3:  $P' = Shake(P, k)$
    **Determining the objective function value for the solution P':**
4:  $f' = 0$
5:  For-Each $u \in V$
6:      If $f(P', u) > f'$
7:          $f' = f(P', u); c_u' = u$
8:      End If
9:  End For-Each
    **Local search:**
10: *Main loop:* While(True)
11:     $f'' = \infty$
12:     For-Each $c_{in} \in V \setminus P'$
            **NULL indicates that no one center is closed**
13:         If $f' > f(P', c_u', c_{in}, NULL)$ or $f' > f(P', c_u', c_{in}, reference(P', c_u'))$
14:             For-Each $c_{out} \in P'$
15:                 $f_{cur} = 0$
16:                 For-Each $u \in V$
17:                     If $f_{cur} < f(P', u, c_{in}, c_{out})$
18:                         $f_{cur} = f(P', u, c_{in}, c_{out}); u_{cur} = u$
19:                     End If
20:                 End For-Each
21:                 If $f'' > f_{cur}$
22:                     $f'' = f_{cur}; in = c_{in}; out = c_{out}; c_u'' = u_{cur}$
23:                 End If
24:             End For-Each
25:         End If
26:     End For-Each
27:     If $f'' < f'$
28:         $f' = f''; c_u' = c_u''; P' = P' \cup \{in\} \setminus \{out\}$
29:     Else
30:         Break *Main loop*
31:     End If
32:   End *Main loop*
      **Jump into a new neighborhood:**
33:   If $f_{value} \geq f'$
34:       $f_{value} = f'; P = P'$
35:       $k = 1$
36:   Else
37:       $k = k \% k_{max} + 1$
38:   End If
39: While CPU_Time() $\leq t_{max}$
40: Return $f_{value}, P$
```

Starting from k = 1 and the predefined solution P, the search continues in a randomly selected new solution P' from the $N_k(P)$ neighborhood (line 3). After initialization of the objective function value f' of the solution P' and identification of the critical user c_u' (lines 4 - 9), i.e. users whose distance to the backup via the reference center is the largest, the local search phase begins (lines 10 - 32). Only c_{in} centers that potentially reduce the distance related to the critical user c_u' are considered (line 13). A pair of (c_{in}, c_{out}) centers is required, where $c_{out} \in P'$ and $c_{in} \in V \setminus P'$, whose exchange yields a new $N_1(P')$ solution with the smallest value of the objective function. The search continues as long as it is possible to find at least one such pair of centers.

If a better or the same solution is found during the local search, the new solution becomes current and the search is reset to k = 1 (lines 33 - 35); otherwise the new solution is discarded and the search continues by jumping into k + 1 neighborhood (line 37). If the maximum value (k_{max}) is exceeded, k is also reset to the initial value. The search ends when the time limit t_{max} is exceeded.

Since the time complexity of auxiliary functions in the worst case is O(p), it is simple to determine the time complexity of the local search iteration (*main loop*) as $O(n * (p + p * n * p)) = O(p^2 n^2)$ in the worst case.

3. Results

The algorithm is implemented in the C++ programming language, and all tests were performed on an Intel Core i7-8700K (3.7GHz) CPU with 32GB RAM. Test examples are taken from the work of Lopez-Sanchez *et al.* (2018), and the obtained results were compared with the hybrid algorithm as the best version of the algorithms proposed in the same paper.

Test instances were generated based on the OR-Library (Beasley, 1990) data set by considering the first *n* nodes from the pmed1-pmed4 and pmed6-pmed8 examples with different *p* values. All test instances are divided into two groups: smaller up to 50 and larger up to 200 nodes.

The proposed algorithm was executed 20 times on each test instance with values $k_{max} = p$ and $t_{max} = 2n$ seconds where *n* is the total number of nodes (users) in a given test example. The results are presented in Tables 1-3.

Table 1

The Results of the Algorithm applied to Smaller Instances Compared to the Results of the Hybrid Algorithm

	P	N	Optimal	Hybrid Value	Time (sec)	Worst Value	AVG Value	Best Value	#Best	Gap Worst vs. Hybrid (%)	Gap AVG vs. Hybrid (%)	Gap Best vs. Hybrid (%)
pmed1	5	10	84	84	0.002	84	84	84	20	0	0	0
pmed1	5	20	120	120	0.034	120	120	120	20	0	0	0
pmed1	10	20	95	95	0.029	95	95	95	20	0	0	0
pmed1	5	30	126	126	0.032	148	131.50	126	15	17.46	4.37	0
pmed1	10	30	95	95	0.316	100	95.25	95	19	5.26	0.26	0
pmed1	5	40	144	144	3.700	154	146.25	144	12	6.94	1.56	0
pmed1	10	40	111	111	13.334	111	111	111	20	0	0	0
pmed1	20	40	89	89	0.229	89	89	89	20	0	0	0
pmed1	10	50	110	111	6.656	115	112.30	110	2	3.60	1.17	**-0.90**
pmed1	20	50	89	89	26.83	91	89.10	89	19	2.25	0.11	0
pmed2	5	10	121	128	0.003	128	124.15	121	11	0	**-3.01**	**-5.47**
pmed2	5	20	147	147	0.014	166	153.65	147	13	12.93	4.52	0
pmed2	10	20	99	99	0.150	99	99	99	20	0	0	0
pmed2	5	30	169	169	0.223	179	169.65	169	18	5.92	0.38	0
pmed2	10	30	110	110	0.849	110	110	110	20	0	0	0
pmed2	5	40	164	164	0.276	164	164	164	20	0	0	0
pmed2	10	40	112	112	5.625	138	124.70	112	6	23.21	11.34	0
pmed2	20	40	96	96	3.589	96	96	96	20	0	0	0
pmed2	10	50	140	140	2.175	145	140.40	140	18	3.57	0.29	0
pmed2	20	50	99	99	8.719	102	99.15	99	19	3.03	0.15	0
pmed3	5	10	77	77	0.002	77	77	77	20	0	0	0
pmed3	5	20	145	145	0.033	167	149.60	145	14	15.17	3.17	0
pmed3	10	20	77	77	0.047	129	87.40	77	**16**	**67.53**	**13.51**	0
pmed3	5	30	157	157	0.094	167	159	157	16	6.37	1.27	0
pmed3	10	30	122	122	0.096	133	122.55	122	19	9.02	0.45	0
pmed3	5	40	157	157	0.190	167	164.05	157	5	6.37	4.49	0
pmed3	10	40	105	105	2.635	125	111.55	105	13	19.05	6.24	0
pmed3	20	40	77	77	2.572	77	77	77	20	0	0	0
pmed3	10	50	125	125	9.641	127	126.30	125	7	1.60	1.04	0
pmed3	20	50	87	87	11.380	87	87	87	20	0	0	0
pmed4	5	10	126	126	0.003	126	126	126	20	0	0	0
pmed4	5	20	139	139	0.023	179	145	139	17	28.78	4.32	0
pmed4	10	20	125	125	0.188	125	125	125	20	0	0	0
pmed4	5	30	173	173	0.086	180	174.40	173	16	4.05	0.81	0
pmed4	10	30	122	122	0.439	122	122	122	20	0	0	0
pmed4	5	40	175	175	0.369	175	175	175	20	0	0	0
pmed4	10	40	122	122	5.590	145	124.05	122	7	18.85	1.68	0
pmed4	20	40	85	85	2.049	85	85	85	20	0	0	0
pmed4	10	50	126	126	14.698	140	130.20	126	14	11.11	3.33	0
pmed4	20	50	91	91	12.589	92	91.70	91	6	1.10	0.77	0
AVG					**3.388**				**16.05**	6.83	1.56	**-0.16**

Table 1 columns represent respectively: the name of the test instance, the number of centers p, the number of users n, the optimal value of the solution, the solution obtained by the hybrid algorithm (Lopez-Sanchez *et al.,* 2018), average time to find the best solution, the worst solution among 20 algorithm executions, the average and the best solution, how many times the best solution was found, gap of the worst, the average and the best solution in relation to the hybrid algorithm solution. The gaps between the obtained solutions and the solutions of the hybrid algorithm were calculated as $\frac{Found\ value - Hybrid\ value}{Hybrid\ value} * 100$. The table contains the test results on smaller test instances, and it was shown that from 20 executions the algorithm managed to find the optimal solution. The optimal solutions were obtained by exact mathematical models from the work of Albared-Sambol *et al.* (2015). The average CPU time required to find the best solution is 3,388s, and an average of 16.05 out of 20 executions identifies the optimal solution. On the other hand, in comparison with the results obtained by the hybrid algorithm, the deviation of the values of the worst and average solutions is noticeable. For example, an instance of pmed3 for p = 10 and n = 20 results in a gap of 67.53% and 13.51% in the worst and average case, respectively, relative to the result of the hybrid algorithm. Nevertheless, execution over that instance returned the optimal solution in 16 out of the 20 executions, so it can be concluded that the algorithm applied to smaller test instances gives acceptable results. The best results in a couple of examples: pmed1 for p = 10 and n = 50, as well as pmed2 for p = 5 and n = 10 improve the solutions of the hybrid algorithm, i.e. return a negative gap of -0.90% and -5.47%. In fact, in the second example, the algorithm as its worst solution identifies the hybrid algorithm solution, so that the average solution with gap of – 3.01% is better.

Table 2
The Results of the Algorithm Applied to Larger Instances compared to the Results of the Hybrid Algorithm

	P	N	Hybrid Value	Time (sec)	Worst Value	AVG Value	Best Value	#Best	Gap Worst vs. Hybrid (%)	Gap AVG vs. Hybrid (%)	Gap Best vs. Hybrid (%)
pmed6	20	150	79	222.75	91	84.60	80	4	15.19	7.09	1.27
pmed6	30	150	71	250.42	85	78.65	74	1	19.72	10.77	4.23
pmed6	50	150	62	240.37	72	63.60	60	2	16.13	2.58	**-3.23**
pmed6	80	150	56	68.05	57	56.05	56	19	1.79	0.09	0
pmed6	20	200	79	224.10	94	87.20	81	3	18.99	10.38	2.53
pmed6	30	200	72	254.00	89	81.35	77	1	23.61	12.99	6.94
pmed6	50	200	68	139.42	82	76.30	70	1	20.59	12.21	2.94
pmed6	80	200	54	224.10	77	64.50	54	2	**42.59**	**19.44**	0
pmed7	20	150	69	226.25	79	71.70	68	1	14.49	3.91	**-1.45**
pmed7	30	150	62	254.15	70	66.65	63	2	12.90	7.50	1.61
pmed7	50	150	59	209.71	66	59.85	59	15	11.86	1.44	0
pmed7	80	150	59	17.47	59	59.00	59	20	0	0	0
pmed7	20	200	73	195.70	88	82.65	75	1	20.55	13.22	2.74
pmed7	30	200	68	222.79	86	76.05	67	1	26.47	11.84	**-1.47**

	P	N	Hybrid Value	Time (sec)	Worst Value	AVG Value	Best Value	#Best	Gap Worst vs. Hybrid (%)	Gap AVG vs. Hybrid (%)	Gap Best vs. Hybrid (%)
pmed7	50	200	63	211.34	70	68.05	63	1	11.11	8.02	0
pmed7	80	200	52	210.42	67	58.40	52	1	28.85	12.31	0
pmed8	20	150	74	244.01	83	78.75	74	2	12.16	6.42	0
pmed8	30	150	61	218.50	83	69.05	63	1	36.07	13.20	3.28
pmed8	50	150	58	263.65	61	58.35	58	16	5.17	0.60	0
pmed8	80	150	58	28.23	58	58.00	58	20	0	0	0
pmed8	20	200	84	222.01	94	91.10	88	5	11.90	8.45	4.76
pmed8	30	200	77	226.27	94	87.25	84	6	22.08	13.31	9.09
pmed8	50	200	68	247.91	88	77.40	68	2	29.41	13.82	0
pmed8	80	200	68	154.88	72	68.20	68	19	5.88	0.29	0
AVG				**199.02**				**6.08**	16.98	7.91	**1.39**

Table 2 presents the results of the algorithm applied to a group of larger test instances. Apart from that it does not contain a column with optimal solutions; the other columns are identical to the columns of Table 1. We did not have optimal solutions for larger instances, so we present only the results of comparison with the results of the hybrid algorithm. The results of the proposed algorithm are not as good as for smaller instances. It is noticeable that the best solution is obtained only for 6.08 out of 20 executions on average and that in most cases the worst and average solution gives a significant gap in relation to the solution of the hybrid algorithm. Differences in gaps between the worst, average, and best solutions, such as in the case of the pmed6 instance for p = 80 and n = 200 of 42.59%, 19.44%, and 0%, respectively, suggest that more effort is needed to stabilize the algorithm. In only 3 out of 24 cases, the algorithm returned the better values (negative gap in the last column) than the hybrid algorithm, while the hybrid algorithm found a better solution for 10 out of 24 test instances. Although the average gap value is 1.39%, it is obvious that the hybrid algorithm has an advantage when treating larger instances. The average time to find the best solution is also large, 199.02s.

Table 3
Comparison with the Results of the Hybrid Algorithm in Case of increased Time Limit t_{max}

	P	N	Hybrid Value	Time (sec)	Worst Value	AVG Value	Best Value	#Best	Gap Worst vs. Hybrid (%)	Gap AVG vs. Hybrid (%)	Gap Best vs. Hybrid (%)
pmed6	20	150	79	496.24	92	83.15	77	1	16.46	5.25	**-2.53**
pmed6	30	150	71	610.05	81	75.75	67	1	14.08	6.69	**-5.63**
pmed6	50	150	62	688.80	63	59.10	56	3	1.61	-4.68	**-9.68**
pmed6	80	150	56	59.43	56	56.00	56	20	0	0	0
pmed6	20	200	79	513.29	91	83.90	80	4	15.19	6.20	1.27
pmed6	30	200	72	428.45	81	79.25	72	1	12.50	10.07	0
pmed6	50	200	68	621.61	81	71.50	62	1	19.12	5.15	**-8.82**
pmed6	80	200	54	721.79	72	53.80	49	3	33.33	-0.37	**-9.26**
pmed7	20	150	69	658.24	78	69.85	68	6	13.04	1.23	**-1.45**
pmed7	30	150	62	602.54	69	65.55	61	1	11.29	5.73	**-1.61**
pmed7	50	150	59	212.89	63	59.20	59	19	6.78	0.34	0
pmed7	80	150	59	25.78	59	59.00	59	20	0	0	0
pmed7	20	200	73	507.34	84	79.85	70	1	15.07	9.38	**-4.11**
pmed7	30	200	68	659.96	84	71.95	67	1	23.53	5.81	**-1.47**
pmed7	50	200	63	594.09	71	65.85	58	1	12.70	4.52	**-7.94**
pmed7	80	200	52	584.48	63	55.55	48	1	21.15	6.83	**-7.69**
pmed8	20	150	74	527.45	84	78.35	74	3	13.51	5.88	0
pmed8	30	150	61	521.21	70	65.45	61	1	14.75	7.30	0
pmed8	50	150	58	301.59	61	58.20	58	18	5.17	0.34	0
pmed8	80	150	58	22.25	58	58.00	58	20	0	0	0
pmed8	20	200	84	556.46	93	89.00	84	2	10.71	5.95	0
pmed8	30	200	77	343.06	88	84.90	77	1	14.29	10.26	0
pmed8	50	200	68	754.32	84	72.10	68	4	23.53	6.03	0
pmed8	80	200	68	216.30	68	68.00	68	20	0	0	0
AVG				**467.82**				6.38	12.41	4.08	**-2.46**

Given the initially poor results for larger examples, we decided to increase the time limit and then again executed the algorithm 20 times for each test instance. By increasing the execution time limit t_{max} to *5n* seconds, significantly better results were obtained, presented in Table 3. It is noticeable that the average time to find the best solution has increased to 467.82s, which is not acceptable. For example, the average execution time of a hybrid algorithm, scaled to the processor we used, is 141.65s. Thus, execution time optimization is required for the algorithm to be used on larger instances of the problem. On the other hand, the proposed algorithm, with a compromise of longer execution time, returns significantly better results than the hybrid algorithm. In as many as 11 out of 24 cases, the algorithm found better solutions than the hybrid algorithm (negative gap in the last column), while the hybrid algorithm returned a better result for only one test instance.

Encouraged by the previous results, we decided to run the algorithm 20 times and over the complete OR-Library test set for $k_{max} = p$ and $t_{max} = 5n$ seconds. The results of the executions are presented in the following table.

Table 4
The Results of the Algorithm applied to Complete OR-Library Test Set

	P	N	Time (sec)	Worst Value	AVG Value	Best Value	#Best	Gap Worst vs. Best (%)	Gap AVG vs. Best (%)
pmed1	5	100	0.50	172	166.30	166	19	3.61	0.18
pmed2	10	100	100.46	147	136.05	135	16	8.89	0.78
pmed3	10	100	87.24	164	153.80	151	15	8.61	1.85
pmed4	20	100	177.84	125	119.35	118	5	5.93	1.14
pmed5	33	100	38.63	85	85.00	85	20	0	0
pmed6	5	200	146.09	111	107.90	107	14	3.74	0.84
pmed7	10	200	390.52	91	86.00	84	2	8.33	2.38
pmed8	20	200	332.01	92	86.85	84	6	9.52	3.39
pmed9	40	200	261.44	75	74.60	71	2	5.63	5.07
pmed10	67	200	9.74	70	70.00	70	20	0	0
pmed11	5	300	168.67	72	70.10	70	19	2.86	0.14
pmed12	10	300	544.26	79	72.75	72	16	9.72	1.04
pmed13	30	300	658.49	65	60.95	52	1	25.00	17.21
pmed14	60	300	515.14	61	60.60	60	8	1.67	1.00
pmed15	100	300	811.39	48	44.95	44	13	9.09	2.16
pmed16	5	400	106.52	57	55.10	55	19	3.64	0.18
pmed17	10	400	791.43	53	49.70	47	4	12.77	5.74
pmed18	40	400	1101.69	55	52.40	50	6	10.00	4.80
pmed19	80	400	1113.12	46	43.50	40	1	15.00	8.75
pmed20	133	400	1079.11	48	43.30	40	4	20.00	8.25
pmed21	5	500	377.36	50	48.80	48	5	4.17	1.67
pmed22	10	500	849.43	58	55.05	52	6	11.54	5.87
pmed23	50	500	1371.95	47	44.40	42	2	11.90	5.71
pmed24	100	500	1196.55	45	38.40	35	1	28.57	9.71
pmed25	167	500	242.45	44	44.00	44	20	0	0
pmed26	5	600	905.53	49	47.95	47	7	4.26	2.02
pmed27	10	600	946.39	43	41.20	40	9	7.50	3.00
pmed28	60	600	20.40	57	57.00	57	20	0	0
pmed29	120	600	1461.31	42	37.25	36	9	16.67	3.47
pmed30	200	600	108.86	40	40.00	40	20	0	0
pmed31	5	700	1057.44	40	36.95	35	7	14.29	5.57
pmed32	10	700	15.29	72	72.00	72	20	0	0
pmed33	70	700	1611.42	37	34.85	33	2	12.12	5.61
pmed34	140	700	72.14	41	41.00	41	20	0	0
pmed35	5	800	927.56	38	36.85	36	4	5.56	2.36
pmed36	10	800	223.52	42	42.00	42	20	0	0
pmed37	80	800	1634.93	38	34.90	33	6	15.15	5.76
pmed38	5	900	1102.41	41	40.20	40	16	2.50	0.50
pmed39	10	900	32.70	74	74.00	74	20	0	0
pmed40	90	900	1394.52	35	32.10	29	1	20.69	10.69
AVG			**599.66**				**10.62**	**7.97**	**3.17**

There have been no results in the literature for the p-next center problem over a complete OR-Library test set, so Table 4 gives only the results for our algorithm. The worst and average solutions were compared to the best solution for each test instance. The solution gap in relation to the best solution was calculated as $\frac{Value - Best\ value}{Best\ value} * 100$. Same as in the case of application over larger instances from the work of Lopez-Sanchez *et al.* (2018), it is noticeable that there is a deviation of the worst and average values (7.97% and 3.17% respectively) in relation to the best solutions, which indicates that it is necessary to further stabilize the algorithm. The best solution is obtained for 10.62 out of 20 executions on average. The average time to find the best solution is also large, 599.66s.

4. Conclusion

The paper discusses the p-next center problem, which is a generalization of the well-known and highly studied p-center problem. The Variable Neighborhood Search algorithm was designed and implemented as a metaheuristic approach to solving the p-next center problem. The solution to the problem is the identification of *p* centers in order to minimize the maximum distance among *n* users to the nearest center plus the distance to its nearest center. It provides as close as possible, not just reference, but also a backup center, capable of serving the user after disabling the reference center.

The proposed VNS algorithm was tested on instances from the literature up to 200 nodes and the obtained experimental results were compared with the results of the hybrid GRASP-VNS algorithm from the work of Lopez-Sanchez *et al.* (2018). The algorithm successfully reproduced optimal solutions over the test instances of smaller dimensions, while significantly larger CPU time is required to solve larger instances of the problem. The proposed algorithm in all but one case identified the same or better solutions compared to the hybrid algorithm, which qualifies it as the most accurate algorithm for solving the p-next center problem.

The examples on which we compared the results are in the domain of small problems, so the plan is to check the results of the algorithm applied to larger problems for which it is certain that solutions cannot be obtained by exact methods. Taking into account the obtained results, in order to do so, it is necessary to optimize the execution time. The algorithm is designed in such a way, that it is easy to extend it to take into account more centers in case the backup center is disabled. The problem is certainly realistic and it would be interesting to try to minimize the maximum distance to the second, third or all other backup centers.

Acknowledgement

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, through Mathematical Institute of the Serbian Academy of Sciences and Arts [grant number OI 174010].

References

Albareda-Sambola, M.; Hinojosa, Y.; Marin, A.; Puerto, J. 2015. When centers can fail: a close second opportunity, *Computers & Operations Research* 62: 145–156.

Beasley, J.E. 1990. OR-Library: distributing test problems by electronic mail, *Journal of the operational research society* 41(11): 1069–1072.

Calik, H.; Tansel, B.C. 2013. Double bound method for solving the p-center location problem, *Computers & operations research* 40(12): 2991–2999.

Daskin, M.S. 2013. Network and discrete location: models, algorithms, and applications, 2nd edn. Wiley, Hoboken. 520p.

Davidovic, T.; Ramljak, D.; Selmic, M.; Teodorovic, D. 2011. Bee colony optimization for the p-center problem, *Computers & Operations Research* 38: 1367-1376. doi: 10.1016 / j.cor.2010.12.002.

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

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

Hakimi, S.L. 1965. Optimum distribution of switching centers in a communication network and some related graph theoretic problems, *Operations Research* 13(3): 462–475.

Hochbaum, D.S.; Shmoys, D.B. 1985. A best possible heuristic for the k-center problem, *Mathematics of Operations Research* 10(2): 180–184.

Ilhan, T.; Pınar, M.C. 2001. An efficient exact algorithm for the vertex p-center problem. Technical report, Department of Industrial Engineering, Bilkent University

Kariv, O.; Hakimi, S.L. 1979. An algorithmic approach to network location problems. Part 1: The p-Centers, *SIAM Journal on Applied Mathematics* 37(3): 513-538.

Lopez, A.; Sánchez-Oro Calvo, J.; Hernández-Díaz, A. 2018. GRASP and VNS for solving the p-next center problem, Computers & Operations Research 104: 295-303. doi: 10.1016 / j.cor.2018.12.017.

Minieka, E. 1970. The m-center problem, *SIAM Rev* 12:138–139.

Mladenovic, N.; Hansen, P. 1997. Variable neighborhood search, *Comput. Operator. Res.* 24 (11): 1097–1100.

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

Pullan, W. 2008. A memetic genetic algorithm for the vertex p-center problem, *Evol Comput* 16:417–436.