Computer Science and Information Systems 20(1):95–115 https://doi.org/10.2298/CSIS210804049R

Solving the P-Second Center Problem with Variable Neighborhood Search

Dalibor Ristic[1,*], Dragan Urosevic[1, 2], Nenad Mladenovic[3], and Raca Todosijevic[4]

[1] School of Computing, Union University, Knez Mihailova 6, 11000 Belgrade, Serbia
dalibor.ristic@outlook.com
[2] Mathematical Institute of the Serbian Academy of Sciences and Arts, Knez Mihailova 36, 11000 Belgrade, Serbia
durosevic@raf.rs
[3] Khalifa University, PO Box 127788, Abu Dhabi, United Arab Emirates
nenadmladenovic12@gmail.com
[4] Polytechnic University of Hauts-de-France, Cedex 9, Valenciennes, France
racatodosijevic@gmail.com

Abstract. The p-center problem is a well-known and highly studied problem pertaining to the identification of p of the potential n center locations in such a way as to minimize the maximum distance between the users and the closest center. As opposed to the p-center, the p-second center problem minimizes the maximum sum of the distances from the users to the closest and the second closest centers. In this paper, we propose a new Variable Neighborhood Search based algorithm for solving the p-second center problem. Its performance is assessed on the benchmark instances from the literature. Moreover, to further evaluate the algorithm's performance, we generated larger instances with 1000, 1500, 2000, and 2500 nodes and instances defined over graphs up to 1000 nodes with different densities. The obtained results clearly demonstrate the effectiveness and efficiency of the proposed algorithm.

Keywords: variable neighborhood method, heuristic algorithms, p-second center problem, combinatorial optimization.

1. Introduction

The p-center problem (pCP) was introduced in 1965 [7] as a discrete optimization problem of identifying p from potential n centers in such a way as to minimize the maximum distance between the users and their closest center. The p-center in the real

[*] Corresponding author

world may appear as a problem of determining locations for the construction of hospitals, so that the distance of the farthest settlement to the hospital closest to it is the smallest possible. The same problem can be applied to the installation of gas stations, fire stations, etc.

The p-center problem is formally defined over an undirected weighted graph G = (V, E), where V is the set of all nodes, i.e. the locations of the centers and the users, while E is the set of all graph edges connecting those locations. The weights of the edges correspond to the distance between their ends. Let *d(i, j)* be the shortest distance between the *i* and *j* nodes in the G graph. If nodes *i* and *j* are not connected, *d(i, j)* is equal to infinity. The solution to the p-center problem is a set of nodes $P \subset V$, of cardinality p, so that the maximum distance from the users to the assigned center is minimized:

$$pCP(V,E) = \min_{\substack{P \subset V,\\ |P|=p}} \max_{i \in V} \left\{ \min_{\substack{j \in P,\\ (i,j) \in E}} d(i,j) \right\} \tag{1}$$

The p-center is an NP-hard problem [11], but it has been known and studied for a long time, so there are many articles and algorithms that deal with the problem. In the literature there are many exact mathematical models and heuristic algorithms that successfully find solutions to the p-center problem. Exact methods, such as [3], [4], [6], [10] and [13], deal with smaller problems, while solutions of larger instances are found by heuristic algorithms. The best-known heuristic algorithms are presented in [5], [9], [15] and [16].

The p-center does not offer a solution to the problem when the assigned center is not able to serve the user. The problem arises when, in the conditions of humanitarian catastrophes, the center becomes overloaded or there is a failure at the center caused by any reason. A simple solution is to assign a backup center to each user in the event of a primary failure. Guided by this idea, Albaredo-Sambola et al. [1] defined the p-next center problem (pNCP) in 2015. The problem of the p-next center is a generalization of the pCP. In the pNCP, it is required to select p from potential n centers in such a way as to minimize not only the maximum distance between the users and the centers closest to them but also the distance between the center and its closest center. Over the same graph G = (V, E) as in the case of the pCP, the p-next center problem is formally defined as:

$$pNCP(V,E) = \min_{\substack{P \subset V,\\ |P|=p}} \max_{i \in V} \left\{ \min_{\substack{j \in P,\\ (i,j) \in E}} d(i,j) + \min_{\substack{k \in P,\\ (j',k) \in E\\ k \neq j' \in \arg\min_{j \in P} d(i,j)}} d(j',k) \right\}, \tag{2}$$

where $arg\, min_{j \in P}\, d(i,j)$ corresponds to node *j* which is closest to node *i*

The G graph is a plane graph with no limits in terms of number of edges and connectivity. The weights of the edges correspond to the Euclidean distances between their nodes and therefore the distances satisfy the triangle inequality.

The p-next center problem, as a generalization of the pCP, is another NP-hard problem. The authors in the paper [1] present a few exact mathematical models as a solution to the problem, which are applicable to smaller instances of the problem. In

2019, Lopez-Sanchez et al. in their work [12] published the first heuristic algorithms for solving the p-next center problem.

It is expected that it is known in advance whether it is necessary to visit the backup center, and therefore, the p-second center problem (pSCP) is defined in response to the potential failure of the primary center [12]. The p-second center problem is a generalization of the p-center problem, in terms of identifying p out of n centers in order to minimize the maximum sum of the distances from the users to the closest and the second closest center. Formally, let G = (V, E) again be an undirected weighted graph, where the weights of the edges are determined by the distance between their ends, V is the set of all nodes, and E is the set of the edges. The centers, as well as other users, represent graph nodes, while *d(i, j)* is the shortest distance between the *i* and *j* nodes, calculated as a result of an algorithm for determining the shortest paths in the graph G. The solution of the p-second center problem is a set of nodes $P \subset V$, of cardinality p, so that the maximum distance from the users ($i \in V$) to the closest center ($j \in P$), plus the distance to the second closest center ($k \in P$) is minimized:

$$pSCP(V,E) = \min_{\substack{P\subset V,\\ |P|=p}} \max_{i\in V} \left\{ \min_{\substack{j\in P,\\ (i,j)\in E}} d(i,j) + \min_{\substack{k\in P,\\ (i,k)\in E,\\ k\neq j}} d(i,k) \right\} \quad \textbf{(3)}$$

The p-second center problem, as an extension of the pCP, is an NP-hard problem. To solve this problem, we propose a heuristic algorithm based on the variable neighborhood search method that includes an efficient local search method to accelerate the convergence to the local optimum. To this end, in the next section, through the description and pseudocode, we present the proposed algorithm. In the third section, we present a modification of the algorithm capable to recognize whether the found solution is optimal. In the fourth section, we present the obtained results of testing the proposed algorithms over a OR-Library [2] set of test instances, as well as two additional test sets generated for testing the algorithm on large instances of the problem. We end the paper with a short summary and an announcement of future work. We also compare the p-second center problem with the p-center and p-next center problems. There is an example graph that illustrates the pCP, the pNCP and the pSCP in Appendix 1.

Briefly, the contributions of our study are:

1. It is interesting to note that the property of finding the next better critical point in the interchange neighborhood lies within a circle whose radius is the current objective function value, which holds for all 3 variants of p-center problem. We proved this property for the p-second center problem.

2. However, for the p-center [15] and the p-next center [17] problems, we must keep track of the first and the second closest centers in the data structure, to assure efficient updating in local search. For the p-second center problem, we prove that the fast interchange move, first proposed by Whitaker in 1983 for solving p-median problem, can be performed by taking track of the third closest center of any user as well. This is another contribution of our study.

3. We included the fast interchange local search into the basic Variable neighborhood search and perform extensive numerical analysis on 40 OR-Library instances with up to 900 facilities and on new generated larger instances.

2. Algorithm

The proposed algorithm for solving the p-second center problem is based on Variable Neighborhood Search metaheuristic (VNS). The VNS was introduced by Mladenovic and Hansen (1997) [14] as a generic framework for building search algorithms. Starting from a predefined current solution, the VNS method continues searching within the randomly selected solution from the appropriate neighborhood of the current solution. The first neighborhood of a solution P, denoted by $N_1(P)$, contains solutions that differ from the solution P in exactly one element. In general, the set of solutions of the k-th neighborhood of the solution P is defined as:

$$N_k(P) = \{P \setminus \{u_1, u_2, \dots, u_k\} \cup \{v_1, v_2, \dots, v_k\} \mid u_1, u_2, \dots, u_k \in P,$$
$$v_1, v_2, \dots, v_k \in (\mathrm{V} \setminus P), u_1 \neq u_2 \neq \ldots \neq u_k, v_1 \neq v_2 \neq \ldots \neq v_k\} \qquad \textbf{(4)}$$

After searching through the N_k neighborhood of the current solution P, in case of finding a better solution P’, the VNS algorithm rejects the previous solution and sets P’ (P = P’) as the new current solution. The search continues within the set of solutions $N_1(P)$. On the other hand, if the search in the set $N_k(P)$ does not produce any better solution, the search continues in the set $N_{k+1}(P)$, where $1 \leq k \leq |P| - 1$. The search ends after k exceeds the maximum allowed value, i.e. |P| in our case.

The p-second center problem is the generalization of the p-center problem. Mladenovic et al. (2004) [15] introduced an efficient VNS algorithm for the p-center problem. To offer a solution to the p-second center problem, we took advantage of the original algorithm from [15] with a simple modification so as to minimize not the maximum distance from the users to the closest center but the maximum sum of the distances to the closest and the second closest centers.

Basically, the VNS algorithm in each of its iterations tries to improve the current solution P. It consists of alternating procedures of random selection of a solution from the $N_k(P)$ neighborhood and a local search of the selected solution in order to find the local optimum. To explain the proposed implementation, let us consider an example of the p-second center problem with n = 15 users and p = 3 centers (Fig. 1). The current solution to the problem is a set of centers P = {1, 2, 11}.

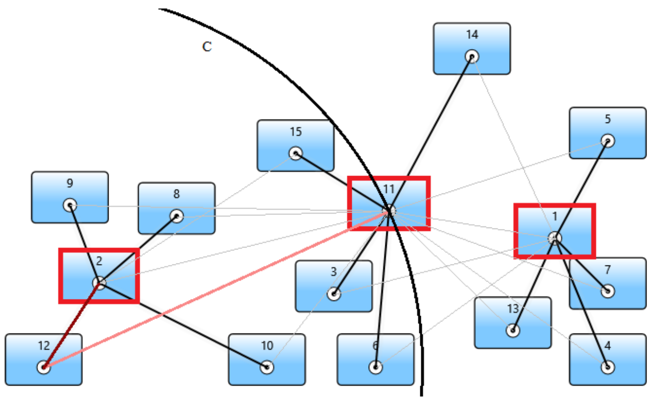

Fig. 1. Example of the p-second center problem with n = 15 and p = 3; the current solution is P = {1, 2, 11}

In Fig. 1, users are connected by darker lines (edges) to their closest, and with lighter to the second closest centers. The distances to the centers and the weights of edges correspond to the lengths of the edges. The user with the largest sum of distances to the appropriate centers is user 12. Let us call it a critical user, $u_c = 12$. The objective function value is just determined by the sum of the distances from critical user to its closest and the second closest center (centers 2 and 11). In order to improve the current function value, it is needed to reduce the distance from the critical user to the closest or/and the second closest center. To that end, it is necessary to find a new center that is closer to the critical user than the second closest center.

Property 1. Let u_c be a critical user, $c_1(u_c)$ its closest, and $c_2(u_c)$ the second closest center in the current solution P, ($c_1(u_c) \in P$, $c_2(u_c) \in P$). Then, if there is a better solution in the neighborhood $N_1(P)$ than the current solution P, the new center c_{in} ($c_{in} \neq c_1(u_c)$) must be closer to the critical user u_c than its previously second closest center $c_2(u_c)$.

Proof. Let f(P) be the function value for the current solution P and $f(u_i)$ the function value for the user u_i; $d(u_i, u_j)$ represents the shortest distance between the u_i and u_j nodes. Then, it applies:

$$f(P) = \max_{i=1,\dots,n} f(u_i) = f(u_c) = d\big(u_c, c_1(u_c)\big) + d\big(u_c, c_2(u_c)\big) \qquad \textbf{(5)}$$

The objective function value is determined by the function value for the critical user. Therefore, in order to reduce the objective function value, it is necessary to include a new center c_{in} into the current solution, so that:

$$d\big(u_c, c_1(u_c)\big) + d(u_c, c_{in}) < d\big(u_c, c_1(u_c)\big) + d\big(u_c, c_2(u_c)\big) \qquad \textbf{(6)}$$

i.e.

$$d(u_c, c_{in}) < d\big(u_c, c_2(u_c)\big) \qquad \textbf{(7)}$$

Corollary. If the critical user is not unique, Property 1 should be applied for each of the critical users, i.e. $(\forall u \in U_c)\Big(d(u, c_{in}) < d\big(u, c_2(u_c)\big)\Big)$, where U_c is a set of all critical users. Otherwise, there is not any solution in the neighborhood $N_1(P)$ better than the current solution P.

Now, we can show how the cardinality of the set of potentially new centers decreases and thus accelerates convergence toward the local minimum. Previous property allows us to reduce the size of the complete *p * (n - p)* neighborhood of P to *p *|N(u*$_c$*)|*, where

$$N(u_c) = \big\{u \,|\, d\big(u_c, c_1(u_c)\big) + d(u_c, u) < f(P)\big\} \qquad \textbf{(8)}$$

Moreover, this size decreases with subsequent iterations and the speed of convergence to a local minimum increases on average. However, the acceleration reduces only the constant in the heuristic's complexity, but not its worst-case behavior.

Based on Property 1, we reduce the search just to the nodes *u* which meet the condition $d(u_c, u) < d(u_c, c_2(u_c))$. In the specific case from Fig. 1, the set of potentially new centers consists of nodes within the circle C (nodes closer to the critical user 12 than the second-closest center). Let us assume that node 3 is chosen as the new center c_{in}. The new center certainly reduces the distance from the critical user 12 to the assigned centers, but whether the objective function value will also be reduced depends on the users who lose one of their centers. If these users keep the sum of distances to the newly assigned centers at a level lower than the previous objective function value, the objective function value will be improved. The *Move* method (Algorithm 1) identifies a center from the currently searched solution that should be replaced with a new one in order to maximally improve the current solution, and also calculates the new objective function value. In fact, instead of checking all *p* possible center exclusions, in the *New center* step, it is calculated only function values when one of the assigned centers is deleted for each of the users. The values are stored in the corresponding elements of the array *z*. Thereafter, the optimal center to be deleted is found based on the *z* value, as the one corresponding to the minimum value of the objective function (the *Best deletion* step). In this way, the time complexity is reduced from O(pn) to O(n) + O(p) ≈ O(n). Finally, in the *Function calculation* step, the new objective function value is calculated. In the worst case, the time complexity of the *Move* method is O(n).

It is assumed that the current solution P represents the first *p* elements of the array x_{cur}, while the last n - p elements contain the rest of the users. The index of the new center is denoted by c_{in}. Additionally, the suggested algorithm uses the following structures:

- *dist(u, v)* – the shortest distance between the *u* and *v* nodes;
- *c1(u)* – the closest center of the user *u*;
- *c2(u)* – the second closest center of the user *u*;
- *c3(u)* – the third closest center of the user *u*;
- *z(v)* – the maximum function value among all users to whom center *v* was assigned either as the closest or the second closest center after the center *v* was removed.

The proposed implementation relies on the algorithms and data structures from the paper [15], provided that the solution is extended with the array c_3 which contains the third closest centers of all users. The additional structure and extensions of the algorithms are conditioned by the nature of the p-second center problem, i.e. by the requirement that the new second closest center is known in advance if one of the two closest centers is removed from the current solution. The extensions do not affect the correctness and efficiency of the algorithm, so the theoretical discussion and properties from the paper [15] remain fully applicable. Note that authors in [15] were analyzing bipartite graph. They did that in order to make clear difference between centers and users. In this paper there is no such assumption, but it does not affect the correctness of the algorithm. The problem defined over the bipartite graph is equivalent to the plane general graph where the weight of edge between the user and center at the same location is 0. Therefore, we just generalized the algorithm and discussion from [15] to plane general graphs and expanded the solution with the auxiliary structure for the third-closest centers to be able to serve the users if the closest centers have failed. The algorithm [15] assigns only one center to each of the users. Therefore, we implemented new algorithm able to find backup center in case that the closest center is broken. Below, we give the pseudocode for the remaining methods that implement the suggested algorithm.

The *Update* method (Algorithm 2) uses the structures *c1*, *c2* and *c3* which represent the lists of the closest and the second and third closest centers of all users. All of them are both input and output values, while c_{in} (the index of the new center) and c_{out} (the index of the center to be deleted from the current solution), along with the current solution x_{cur} represent only input values. The users in the current solution are iterated one by one and if any of their closest centers is deleted (c_{out}) or the new center c_{in} becomes one of the closest, corresponding *c1*, *c2* and *c3* arrays will be updated. The worst-case complexity of the method *Update* is O(n log n) when a heap data structure is used for updating the third closest centers.

Algorithm 1. 1-interchange move in the context of the p-second center problem

```
Move($x_{cur}$, $c_{in}$, c1, c2, c3)
Initialization:
Set $z(x_{cur}(i)) \leftarrow 0$ for all $i = 1, ..., p$

New center:
$in \leftarrow x_{cur}(c_{in})$
For Each $user = x_{cur}(1), ..., x_{cur}(n)$
    If $dist(user, in) < dist(user, c2(user))$
        **in as a new closest or second-closest center**
        $z(c1(user)) \leftarrow max(dist(user, in) + dist(user, c2(user)), z(c1(user)))$
    Else
        **user keeps the same centers**
        $z(c1(user)) \leftarrow max(min(dist(user, in), dist(user, c3(user))) + dist(user, c2(user)),$
                    $z(c1(user)))$
        $z(c2(user)) \leftarrow max(min(dist(user, in), dist(user, c3(user))) + dist(user, c1(user)),$
                    $z(c2(user)))$
    End If
End For Each

Best deletion:
$min \leftarrow \infty$
For Each $i = \{1, ..., p\}$
    If $min > z(x_{cur}(i))$
        $min \leftarrow z(x_{cur}(i))$
        $c_{out} \leftarrow i$
    End If
End For Each

Function calculation:
$f_{cur} \leftarrow 0$
$out \leftarrow x_{cur}(c_{out})$
For Each $user = x_{cur}(1), ..., x_{cur}(n)$
    If $c1(user) = out$
        $c1 \leftarrow c2(user)$
        $c2 \leftarrow c3(user)$
    Else
        $c1 \leftarrow c1(user)$
        $c2 \leftarrow c2(user)$ if $c2(user) \neq out$ else $c3(user)$
    End If
    $f \leftarrow dist(user, c1) + dist(user, in)$ if $dist(user, in) < dist(user, c2)$
       else $dist(user, c1) + dist(user, c2)$
    $f_{cur} \leftarrow max(f, f_{cur})$
End For Each

Return $f_{cur}$, $c_{out}$
```

Algorithm 2. Updating the first, the second and the third-closest center

```
Update($x_{cur}$, $c_{in}$, $c_{out}$, c1, c2, c3)
in ← $x_{cur}(c_{in})$
out ← $x_{cur}(c_{out})$
For Each user = $x_{cur}(1), \ldots, x_{cur}(n)$
    **for users whose center is deleted, find new one**
    If c1(user) = out
        If dist(user, in) ≤ dist(user, c2(user))
            c1(user) ← in
        Else
            c1(user) ← c2(user)
            If dist(user, in) ≤ dist(user, c3(user))
                c2(user) ← in
            Else
                c2(user) ← c3(user)
                **find third closest center for the user**
                c3(user) ← select center
                          from $\{x_{cur}(1), \ldots, x_{cur}(p)\} \cup \{in\} \setminus \{c1(user), c2(user), out\}$
                          where d(user, center) is minimum
            End If
        End If
    Else
        If c2(user) = out
            If dist(user, in) ≤ dist(user, c1(user))
                c2(user) ← c1(user)
                c1(user) ← in
            Else
                If dist(user, in) ≤ dist(user, c3(user))
                    c2(user) ← in
                Else
                    c2(user) ← c3(user)
                    **find third closest center for the user**
                    c3(user) ← select center
                              from $\{x_{cur}(1), \ldots, x_{cur}(p)\} \cup \{in\} \setminus \{c1(user),c2(user), out\}$
                              where d(user, center) is minimum
                End If
            End If
        Else
            If dist(user, in) ≤ dist(user, c1(user))
                c3(user) ← c2(user)
                c2(user) ← c1(user)
                c1(user) ← in
            Else
                If dist(user, in) ≤ dist(user, c2(user))
                    c3(user) ← c2(user)
                    c2(user) ← in
                Else
                    If dist(user, in) ≤ dist(user, c3(user))
                        c3(user) ← in
                    Else If c3(user) = out
                        **find third closest center for the user**
                        c3(user) ← select center
                              from $\{x_{cur}(1), \ldots, x_{cur}(p)\} \cup \{in\} \setminus \{c1(user), c2(user), out\}$
                              where d(user, center) is minimum
                    End If
                End If
            End If
        End If
    End If
End For Each
Return c1, c2, c3
```

Algorithm 3. The vertex substitution local search for the p-second center problem

```
LocalSearchVertexSubstitution($x_{cur}$, c1, c2, c3, $u_c$, $f_{cur}$)
Main loop:
While True
    $f' \leftarrow \infty$
    For Each in = p +1, ..., n
        Find the optimal objective function value improvement within $N_1(x_{cur})$ neighborhood:
        If $d(u_c, x_{cur}(in)) < d(u_c, c2(u_c))$
            f, out $\leftarrow$ Move($x_{cur}$, in, c1, c2, c3)
            If $f < f'$
                $f' \leftarrow f$
                $c_{in} \leftarrow in$
                $c_{out} \leftarrow out$
            End If
        End If
    End For Each

    If $f_{cur} \leq f'$
        There was not found improvement in the neighborhood:
        Break Main loop
    End If

    Update($x_{cur}$, $c_{in}$, $c_{out}$, c1, c2 ,c3)
    $f_{cur} \leftarrow f'$
    $x_{cur}(c_{in}) \leftrightarrow x_{cur}(c_{out})$
    $u_c \leftarrow$ select user from $x_{cur}(1), \ldots, x_{cur}(n)$
        where d(user, c1(user)) + d(user, c2(user)) is maximum
End While
Return $x_{cur}$, $u_c$, $f_{cur}$
```

The Local Search Vertex Substitution method (Algorithm 3), relying on Property 1, accelerates the convergence towards the local optimum. However, it reduces the search space and constant complexity factor, but not the time complexity in the worst case. The complexity of one iteration of the *Main loop* in the worst case remains $O(n^2) + O(n \log n) + O(n) \approx O(n^2)$. The input values are the current solution, the c1, c2 and c3 arrays, the critical user and the current objective function value. The local search method, using the *Move* method, iteratively finds the optimal pair of centers to be exchanged, which results in the largest reduction in the objective function value. The method is executed as long as it is possible to find such a pair of centers. In the end, the method provides a new current solution, a new critical user and a new objective function value.

After the iteration of the proposed VNS implementation, i.e. of the previously described algorithms, over the problem from Fig. 1, we obtain a new current solution P = {1, 2, 3} (Fig. 2). Center 3 is included in the solution, and center 11 is deleted. The new critical user is $u_c = 14$, the center closest to it is center 1, while the second closest center is center 3. The objective function value is reduced, i.e. d(14, 1) + d(14, 3) < d(12, 2) + d(12, 11).

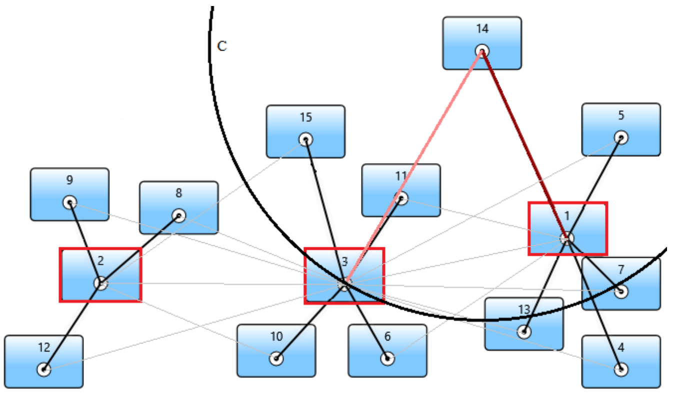

Fig. 2. Example of the p-second center problem with n = 15 and p = 3; the new solution is P = {1, 2, 3}

The pseudocode for the VNS algorithm for the p-second center problem is given in Algorithm 4. The initial value of the variable k is 1. This means that a new solution is searched in the N_1 neighborhood of the initially current solution. In the *Shaking operator* step, the new solution is selected from the N_k neighborhood of the current solution. Same as in the local search method, based on Property 1, the size of the neighborhood N_k set is reduced. It is selected one of the solutions which contain a new center that is closer to the critical user u_c* than the previous second closest center $c2(u_c*)$. After that, the chosen solution x_{cur} becomes the current solution for the local search method (the *Local search* step). If the local search finds an equal or a better solution than x_{cur}, the new solution is set as the “currently optimal” x_{opt}, and the search process restarts, $k = 1$. Otherwise, the value of k is incremented and the search process continues within the set N_{k+1}. If k has reached the maximum value k_{max}, k is set to 1 and the search process is restarted again. The *Main step* is repeated until the maximum allowed execution time t_{max} is reached. The algorithm returns the best solution found during the search process.

Algorithm 4. Shaking procedure for the p-second center problem

```
VariableNeighborhoodSearch($k_{max}$, $t_{max}$)
Initialization:
 Randomly initialize $x_{opt}$; according to $x_{opt}$ initialize arrays c1, c2 and c3, $f_{opt}$, $u_c$*;
 copy initial solution into the current one, i.e., copy $f_{opt}$, $x_{opt}$, c1, c2, c3 and $u_c$* into $f_{cur}$, $x_{cur}$, $c1_{cur}$, $c2_{cur}$,
 $c3_{cur}$ and $u_{cur}$* respectively.
Repeat the Main step until the stopping condition is met (e.g., time $\leq t_{max}$ )
Main step:
    $k \leftarrow 1$
    While $k \leq k_{max}$
        Shaking operator:
        **generate a solution at random from kth neighborhood**
        For Each $j = 1, ..., k$
          1. Take center $c_{in}$ to be inserted at random if $d(u_{cur}*, x_{cur}(c_{in})) < d(u_{cur}*, c2_{cur}(u_{cur}*))$;
          2. Find center $c_{out}$ to be deleted at random;
          3. Update $x_{cur}$, $c1_{cur}$, $c2_{cur}$ and $c3_{cur}$,i.e., execute:
             Update($x_{cur}$, $c_{in}$, $c_{out}$, $c1_{cur}$, $c2_{cur}$, $c3_{cur}$)
             $x_{cur}(c_{in}) \leftrightarrow x_{cur}(c_{out})$
          4. Update $f_{cur}$ and $u_{cur}$* according to $x_{cur}$, $c1_{cur}$, $c2_{cur}$ and $c3_{cur}$
        End For Each
        Local search:
        If any potentially better solution found
            $x_{cur}$, $u_{cur}*$, $f_{cur}$ ← LocalSearchVertexSubstitution($x_{cur}$, c1, c2, c3, $u_{cur}*$, $f_{cur}$)
            Move or not:
            If $f_{cur} \leq f_{opt}$
                **Save current solution as the optimal; return to $N_1$**
                $x_{opt}$ ← $x_{cur}$; $f_{opt}$ ← $f_{cur}$; $u_c*$ ← $u_{cur}*$; c1 ← $c1_{cur}$; c2 ← $c2_{cur}$; c3 ← $c3_{cur}$
                k ← 1
            Else
                ** There was not found better solution; change the neighborhood**
                $x_{cur}$ ← $x_{opt}$; $f_{cur}$ ← $f_{opt}$; $u_{cur}*$ ← $u_c*$; $c1_{cur}$ ← c1; $c2_{cur}$ ← c2; $c3_{cur}$ ← c3
                k ← k + 1
            End If
        End If
    End While
Return $x_{opt}$, $f_{opt}$
```

3. Quasi-Exact Algorithm

Property 2. Let u_c be a critical user, $c_1(u_c)$ its closest and $c_2(u_c)$ the second closest center in the current solution P, ($c_1(u_c) \in P$, $c_2(u_c) \in P$). Then, if in the set of potentially new centers there is no center c_{in} that is closer to the critical user u_c than its second closest center $c_2(u_c)$, the current solution P is the optimal solution for the p-second center problem.

Proof. Based on Property 1, it applies $d(u_c, c_{in}) < d(u_c, c_2(u_c))$. If it is not possible to find a new center c_{in} that satisfies previous inequality, it means that it is not possible to improve the current solution, i.e. the current solution is the optimal solution to the p-second center problem.

Corollary. If there is not any center to be opened in order to improve the objective function value related to the critical user u_c, the critical user has already been selected to be a center in the current solution P.

Property 2 can be considered as a generalization of Property 1, i.e. if there is a better solution P' than the current solution P, the solution P' has to contain a new center c_{in} ($c_{in} \notin P$) which is closer to the critical user u_c than the second closest center $c_2(u_c)$ ($c_2(u_c) \in P$).

Based on Property 2, we propose a modification of the VNS algorithm from the previous section, which might be able to recognize the exact solution to the p-second center problem. The idea is, to check if there is at least one center that potentially improves the current solution. If there is not any, the execution is stopped and the algorithm returns the current solution as the optimal one (Algorithm 5).

As future work, it would be interesting to try to accelerate the convergence towards an optimal solution by searching the history of the previous solutions. For example, the solution P1 = {1, 2, 11} in Fig. 1 is improved by adding center 3 and, after deleting center 11, it becomes {1, 2, 3}. The set of all potentially new centers consists of the centers within the circle C, i.e. $C1_{in}$ = {3, 6, 8, 9, 10, 12, 15}. So, the solution {1, 2} is expanded by one of the centers from the $C1_{in}$ set. In Fig. 2, the current solution is P2 = {1, 2, 3}, and the set of potentially new centers $C2_{in}$ = {5, 11, 14, 15}. In case of deleting node 3, the current solution might be expanded by centers from the set $C2_{in} \cap C1_{in} = \{15\}$, which reduces the cardinality of the search set from 4 to 1. A new structure can be used to store the previous solutions, e.g. a tree that stores in its leaves a set of all potential centers that can extend the solution defined by nodes on the path from the root to that particular leaf of the tree.

Algorithm 5. Shaking procedure for the p-second center problem

```
VariableNeighborhoodSearch($k_{max}$, $t_{max}$)
Initialization:
…
Repeat the Main step until the stopping condition is met (e.g., time $\leq t_{max}$ )
Main step:
    $k \leftarrow 1$
    While $k \leq k_{max}$
    Shaking operator:
    **generate a solution from kth neighborhood**
    $count \leftarrow 0$
    For Each $j = 1, ..., k$
        $c_{in} \leftarrow$ find center at random from $x_{cur}(p + 1), \dots, x_{cur}(n)$
                where $d(u_{cur}^*, x_{cur}(center)) < d(u_{cur}^*, c2_{cur}(u_{cur}^*))$

        If $c_{in}$ not found
            If $count = 0$
                **the optimal solution has been found**
                Return $x_{opt}, f_{opt}$
            End If
        Else
          5. Increment count
          6. Find center to be deleted ($c_{out}$) at random;
          7. Update $x_{cur}$, $c1_{cur}$, $c2_{cur}$ and $c3_{cur}$,i.e., execute:
             Update($x_{cur}$, $c_{in}$, $c_{out}$, $c1_{cur}$, $c2_{cur}$, $c3_{cur}$)
             $x_{cur}(c_{in}) \leftrightarrow x_{cur}(c_{out})$
          8. Update $f_{cur}$ and $u_{cur}^*$ according to $x_{cur}$, $c1_{cur}$, $c2_{cur}$ and $c3_{cur}$
      End If
  End For Each
  Local search:
  …
  End While
```

4. Results

The algorithm was implemented in the C++ programming language, and all tests were performed on an Intel Core i7-8700K (3.7 GHz) CPU with a 32 GB RAM configuration. For testing purposes, we downloaded an OR-Library [2] data set containing 40 test instances with 100 to 900 nodes and p between 5 and 200 (not more than n/3, where n is the number of nodes). Additionally, we generated two data sets with larger test instances. The first contains 44 instances with 1000 to 2500 nodes and p between 5 and 200. The second contains 48 instances (500–1000 nodes) defined over graphs with different densities (50%–80%) and p between 5 and 200.

Each of the proposed algorithms was executed 20 times on each testing instance, always starting from a different initial solution. Different combinations of the parameters $k_{max} = p/4$, $k_{max} = p/2$, $k_{max} = p$, as well as $t_{max} = n$ and $t_{max} = 2n$ were tested. It turned out that the results were slightly better with higher values of the parameter k_{max}. On the other hand, the algorithm usually found the best solution much before the execution time limit expired. Therefore, we decided to present the results only for $k_{max} = p$ and $t_{max} = n$ seconds. The summary results are presented in the following tables, while the detailed results are available in Appendix 2.

4.1. Test results over the OR-Library instances

Table 1 shows the results obtained for original OR-Library test instances. The first column of the table contains the name of the instance, the next three columns represent the value of p (number of centers), n (number of users) and m (number of graph edges). The columns "Best Value", "AVG Value" and "Worst Value" show, respectively, the best, average, and worst solution value that the algorithm found during the 20 executions. The "Time" column (or time-to-target) shows the average time in seconds that was needed to find the best solution for the first time. "Time" does not represent the total execution time. The algorithm is executed until the time limit is reached, i.e. *n* seconds. The last column "#Best" shows the number of times the algorithm found the best solution during the 20 executions. Also, in Appendix 2 we included additional columns containing percentage gaps of the average and the worst solution compared to the best known solution presented in the column "Best(-Known) Value". Since OR-Library instances have not yet been used to test the p-second center problem, we take the best solutions found by our algorithm as the best known solutions.

Table 1. Results for multi-executed OR-Library test instances

	P	N	M	Best Value	AVG Value	Worst Value	Time	#Best
pmed1-pmed5	5-33	100	200	193.80	193.80	193.80	3.30	20.00
pmed6-pmed10	5-67	200	800	120.00	120.02	120.20	4.33	19.60
pmed11-pmed15	5-100	300	1800	83.80	83.80	83.80	2.23	20.00
pmed16-pmed20	5-133	400	3200	65.00	65.00	65.00	34.49	20.00
pmed21-pmed25	5-167	500	5000	58.60	58.61	58.80	48.15	19.80
pmed26-pmed30	5-200	600	7200	56.00	56.00	56.00	35.03	20.00
pmed31-pmed34	5-140	700	9800	53.00	53.00	53.00	19.83	20.00
pmed35-pmed37	5-80	800	1280	51.67	51.83	52.00	149.49	16.67
pmed38-pmed40	5-90	900	16200	54.67	54.98	55.00	29.12	13.67
AVG							**31.32s**	**19.20**

Based on the results, we noticed that most of the smaller instances were solved very quickly. In merely 3 out of 15 cases for n $\leq$ 300, the average time-to-target was more than 4 seconds. As the size of the test instance increases, so does the time needed to find the best solution. The average time-to-target over a complete test set is 31.32 seconds. On the other hand, in terms of the algorithm stability and solution quality, the best-known solution was not found by each of the 20 executions only for 4 out of 40 instances. Moreover, in just two examples (pmed37 and pmed40), the best solution was found less than eighteen times in 20 executions. In the case of the pmed40 instance, only once the best solution was found, but in all other executions a solution was found the value of which is higher by only 1 than the best value. The algorithm found the best known solution on average in 19.20 out of 20 cases or in 96% of the cases. Regarding the deviation of the average and worst solutions, from Table 6 (Appendix 2) it can be concluded that there are no significant differences between the worst/average and best values, only 0.23% and 0.12% on average, respectively.

To verify the quality of our sophisticated local search method, we applied a simple local search method and tested the VNS algorithm with the same parameters and test data. Unlike the proposed solution, the simple local search method does not filter the centers that do not satisfy Property 1. The method only searches for centers that improve the current solution, as shown in Algorithm 6.

Algorithm 6. The simple local search method

```
SimpleLocalSearch($x_{cur}$)
Main loop:
While True
    $P \leftarrow [x_{cur}(1), ..., x_{cur}(p)]$
    $(c_{in}, c_{out}) \leftarrow$ select (user, center)
                where user in $[x_{cur}(p+1), ..., x_{cur}(n)]$ and
                    center in $[x_{cur}(1), ..., x_{cur}(p)]$ and
                    $P \cup \{c_{in}\} \setminus \{ c_{out} \}$ is better than P
    If $(c_{in}, c_{out})$ found
        Exchange($x_{cur}(c_{in}), x_{cur}(c_{out})$)
    Else
        Break Main loop
    End If
Return $x_{cur}$
```

Algorithm 6 was also executed 20 times and the results are presented in Table 2. For the VNS which uses a simple local search algorithm, we show the best solution obtained in 20 executions (the “Best Found Value” column) and the average time to find the best solution (the “Time” column). The “#Best-Known” column contains the number of algorithm executions with a simple local search method that resulted in finding the best known solution, i.e. the best solution found by the algorithm with a sophisticated local search. Finally, the last column gives the percentage deviation of the best solution found by the algorithm with a simple local search method from the best-known solution. The percentage gap is calculated as $\frac{Best\ found - Best\ known}{Best\ known} * 100$.

Table 2 and Table 7 in Appendix 2 show that a VNS with a simple local search method is not capable of providing satisfactory results. The algorithm did not find the best known solution for 14 out of 40 instances, having found the best known solution only in 8.43 out of 20 cases on average. It turns out that the value of the best found solution of the VNS algorithm with a sophisticated local search method is better by

5.47% on average. Moreover, several instances, such as pmed19, pmed24, pmed33, pmed37 and pmed40, resulted in a significantly larger deviation from the best known solution. Also, the average time to find the best solution increased almost eight times, for up to 246.92 seconds. All these findings point to the advantages of a sophisticated local search algorithm, i.e. much better solutions for a significantly less CPU time.

Table 2. VNS with simple local search

	P	N	Best-Known Value (1)	Best-Found Value (2)	Time	#Best-Known	Gap $\left[\frac{(2)-(1)}{(1)} * 100\right]$
pmed1-pmed5	5-33	100	193.80	193.80	16.15	16.80	0.00
pmed6-pmed10	5-67	200	120.00	120.20	64.97	11.60	0.24
pmed11-pmed15	5-100	300	83.80	86.00	183.92	5.60	3.80
pmed16-pmed20	5-133	400	65.00	68.60	260.17	6.00	7.72
pmed21-pmed25	5-167	500	58.60	62.00	337.51	6.60	8.41
pmed26-pmed30	5-200	600	56.00	57.40	340.79	5.40	3.63
pmed31-pmed34	5-140	700	53.00	55.25	328.85	10.25	6.43
pmed35-pmed37	5-80	800	51.67	55.67	487.02	1.67	12.12
pmed38-pmed40	5-90	900	54.67	58.33	360.99	10.33	12.64
AVG					**246.92s**	**8.43**	**5.47%**

Table 3 contains the results of the execution of the quasi-exact algorithm (Algorithm 5) over all the OR-Library test instances. Compared to the previous ones, the table has been expanded with the “Exact Value” column, which shows whether the exact solution has been found. The algorithm was also executed 20 times with the same parameter values (k_{max} = p and t_{max} = n seconds).

Table 3 shows that the quasi-exact algorithm, despite reducing on average the time for finding the best solution, is not as effective as the initial algorithm. It found the best solution in 9.63 out of 20 executions on average. There were several instances (pmed18, pmed19, pmed23 and pmed40) for which no best-known solution was found. As for the best solutions, the initial algorithm is averagely more successful only for 0.24%. On the other hand, the algorithm managed to identify optimal solutions for 12 out of 40 instances from OR-Library test set, which is indicated in the last column of Table 3. In the column “#Best-Known” is reported how many times the algorithm found the best solution. It is important to note that mostly the larger instances, i.e. problems with a greater number of centers (higher p values), were solved exactly. In case of higher p values, it is more likely that there is a center which is a critical user at the same time and it is not possible to find a closer backup center to be included into the current solution in order to reduce the objective function value (Corollary of Property 2).

Table 3. Results of the quasi-exact VNS algorithm for multi-executed OR-Library test instances

	P	N	Best-Known Value (1)	Best-Found Value (2)	Time	#Best-Known	Gap $\left[\frac{(2)-(1)}{(1)} * 100\right]$	Exact Value
pmed1	5	100	268	268	0.02	3	0	
pmed2	10	100	220	220	0.11	4	0	
pmed3	10	100	208	208	0.09	3	0	
pmed4	20	100	163	163	0.07	1	0	
pmed5	33	100	110	110	0.02	16	0	
pmed6	5	200	180	180	0.23	10	0	
pmed7	10	200	143	143	0.51	3	0	
pmed8	20	200	122	122	0.48	2	0	
pmed9	40	200	85	85	0.20	3	0	
pmed10	67	200	70	70	0.10	20	0	✓
pmed11	5	300	125	125	0.52	16	0	
pmed12	10	300	112	112	1.20	5	0	
pmed13	30	300	78	78	1.00	3	0	
pmed14	60	300	60	60	0.63	1	0	✓
pmed15	100	300	44	44	0.36	20	0	✓
pmed16	5	400	98	98	1.61	16	0	
pmed17	10	400	83	83	4.26	9	0	
pmed18	40	400	62	63	2.70	0	1.61	
pmed19	80	400	42	43	1.51	0	2.38	
pmed20	133	400	40	40	0.81	20	0	✓
pmed21	5	500	85	85	5.00	14	0	
pmed22	10	500	80	80	20.38	3	0	
pmed23	50	500	49	50	6.17	0	2.04	
pmed24	100	500	35	35	2.37	1	0	
pmed25	167	500	44	44	1.48	20	0	✓
pmed26	5	600	80	80	20.52	5	0	
pmed27	10	600	67	67	20.04	7	0	
pmed28	60	600	57	57	2.86	20	0	✓
pmed29	120	600	36	36	3.02	20	0	✓
pmed30	200	600	40	40	3.06	20	0	✓
pmed31	5	700	64	64	7.32	20	0	
pmed32	10	700	72	72	4.58	20	0	✓
pmed33	70	700	35	35	16.87	8	0	
pmed34	140	700	41	41	4.50	20	0	✓
pmed35	5	800	64	64	66.86	6	0	
pmed36	10	800	58	58	50.63	8	0	
pmed37	80	800	33	33	24.95	2	0	✓
pmed38	5	900	61	61	32.37	16	0	
pmed39	10	900	74	74	9.82	20	0	✓
pmed40	90	900	29	30	18.08	0	3.45	
AVG					8.43s	9.63	0.24%	**Total 12**

4.2. Test results over larger instances

Encouraged by the obtained results, in order to further assess the performance of the algorithm, we generated test instances with 1000, 1500, 2000, and 2500 nodes, as well as new instances defined over the graphs with up to 1000 nodes with various density values. New test instances are generated as random k-regular graphs with predefined node and edge count. The initial VNS algorithm was again executed 20 times with the same parameter values ($k_{max} = p$ and $t_{max} = n$ seconds) over the new test examples and the results are presented in the following tables and Appendix 2.

Table 4. Results for large test instances

	P	N	M	Best Value	AVG Value	Worst Value	Time	#Best
rndkreg1-rndkreg11	5-200	1000	50000	14.82	14.90	15.00	82.85	18.36
rndkreg12-rndkreg22	5-200	1500	112500	12.45	12.56	12.64	183.81	17.82
rndkreg23-rndkreg33	5-200	2000	200000	11.64	11.64	11.64	235.01	20.00
rndkreg34-rndkreg44	5-200	2500	312500	10.27	10.51	10.64	576.65	15.18
AVG							**269.58s**	**17.84**

Table 4 shows the execution results of the larger test instances and it is noticeable that the algorithm was not as successful as in other instances. It found the best solution in 17.84 out of 20 executions on average. There have been several instances (rndkreg13, rndkreg34 and rndkreg38) for which the best solutions were found only three or fewer times, while the absolute deviation of the worst and best solutions was not higher than 1. The average time-to-target was 269.58 seconds.

In addition to the previously explained columns, Table 5 contains a new column ("Density"), which shows density of the graph over which the test instance is defined.

Based on the results from Table 5, and taking into account the slightly larger instances of the problem, the average time-to-target increased to 43.07 seconds as compared to the OR-Library instances. The average number of finding the best solution dropped to 18.46 out of 20 executions. On the other hand, the density of the graph did not affect the efficiency of the algorithm. This was to be expected, given that the algorithm does not take into account the number of graph edges, but generates new search solutions from appropriate neighborhoods by a simple replacement of nodes (centers). There were only 8 (out of 48) instances for which the best solution was not found in each of the 20 executions, but the absolute deviation of the worst and the best solution in all these cases was only 1**.**

We also compared the suggested algorithm for the pSCP with the results of the algorithms for the p-center [15], the p-next center [17] and the p-median [8] problems. The results are reported in Appendix 3.

Table 5. Results for test instances defined over graphs with different densities

	P	N	Density	Best Value	AVG Value	Worst Value	Time	#Best
rnddnskreg1-rnddnskreg4	5-200	500	50.10	8.00	8.00	8.00	13.07	20.00
rnddnskreg5-rnddnskreg8	5-200	500	60.12	7.00	7.00	7.00	4.35	20.00
rnddnskreg9-rnddnskreg12	5-200	500	80.16	6.00	6.23	6.25	17.13	15.50
rnddnskreg13-rnddnskreg16	5-200	600	50.08	7.25	7.36	7.75	67.27	17.75
rnddnskreg17-rnddnskreg20	5-200	600	60.10	7.00	7.00	7.00	6.45	20.00
rnddnskreg21-rnddnskreg24	5-200	600	80.13	5.75	5.99	6.25	51.02	15.25
rnddnskreg25-rnddnskreg28	50	800	50.06	6.75	6.83	7.00	97.05	18.50
rnddnskreg29-rnddnskreg32	5-200	800	60.08	6.50	6.50	6.50	37.24	20.00
rnddnskreg33-rnddnskreg36	5-200	800	80.10	6.00	6.00	6.00	28.90	20.00
rnddnskreg37-rnddnskreg40	5-200	1000	50.05	6.50	6.56	6.75	105.35	18.75
rnddnskreg41-rnddnskreg44	5-200	1000	60.06	6.50	6.50	6.50	27.76	20.00
rnddnskreg45-rnddnskreg48	5-200	1000	80.08	5.50	5.71	5.75	61.31	15.75
AVG			63.43%				**43.07s**	**18.46**

5. Conclusions

The paper discusses the p-second center problem, which is a generalization of the well-known and highly studied the p-center problem. The algorithm based on Variable Neighborhood Search method has been designed and implemented as a heuristic approach to problem solving. A solution to the p-second center problem represents the identification of the p centers for the purpose of minimizing the maximum distance from the n users to the closest and the second closest centers.

The proposed algorithm was tested on OR-Library instances from the literature with up to 900 nodes, and the obtained experimental results confirmed the high efficiency of the algorithm in a reasonable amount of time. The algorithm found the best solution on average in 19.20 out of 20 cases or in 96% of the cases. The paper also presents a modification of the initially proposed algorithm as a VNS implementation that is capable of identifying the optimal solutions to the p-second center problem. It was tested over the same OR-Library test set. It turned out that the quasi-exact algorithm on average yields worse results as compared to the initial algorithm, but it managed to identify 12 exact solutions out of 40 OR-Library instances.

To confirm the efficiency of the original algorithm, we also generated larger test instances with 1000, 1500, 2000, and 2500 nodes as well as instances defined over graphs of varying densities (50%–80%). It was shown that the density of the graph did not affect the efficiency of the algorithm. The best solution was found for 40 out of 48 instances in each of the 20 executions of the algorithm, or in 92.29% of the cases on average. For larger instances of the problem (up to n=2500), the algorithm was not as successful, but it was still stable. It found the best solution in 89.20% executions with an absolute deviation of the worst solution being not higher than 1.

The proposed algorithm successfully solves the p-second center problem, but it would certainly be interesting to generalize the problem by taking into account more than 2 centers, i.e. apply the VNS heuristic to the p-α-closest center problem, where it would be necessary to minimize the sum of the distances to α closest centers. The proposed quasi-exact algorithm for the p-second center problem showed a 30% success rate in identifying exact solutions. As the topic of a future paper, it would be challenging to try to increase the success rate of the algorithm, for instance, by tracking the search history and comparing the current solution with potential extensions of the previous solutions.

Acknowledgment. This work has been partially supported by the Serbian Ministry of Education, Science and Technological Development through Mathematical Institute of the Serbian Academy of Sciences and Arts. This research has also been partially supported by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan, Grant No. BR10965172.

References

1. Albareda-Sambola, M., Hinojosa, Y., Marín, A., Puerto, J.: When centers can fail: a close second opportunity. Computers & Operations Research, 62, 145–156. (2015)
2. Beasley, J. E.: OR-Library: distributing test problems by electronic mail. Journal of the operational research society, 41 (11), 1069–1072. (1990)
3. Calik, H., Tansel, B. C.: Double bound method for solving the p-center location problem. Computers & Operations Research, 40, 2991–2999. (2013)
4. Daskin, M. S.: Network and discrete location: models, algorithms, and applications, 2nd edn. Wiley, Hoboken. (2013)
5. Davidovic, T., Ramljak, D., Selmic, M., Teodorovic, D.: Bee colony optimization for the p-center problem, Computers and Operations Research, 38, 1367–1376. (2011)
6. Elloumi, S., Labbé, M., Pochet, Y.: A new formulation and resolution method for the p-center problem. INFORMS Journal on Computing, 16, 84–94. (2004)
7. Hakimi, S. L.: Optimum distribution of switching centers in a communication network and some related graph theoretic problems. Operations Research, 13 (3), 462-475. (1965)
8. Hansen, P., Mladenovic, N.: Variable neighborhood search for the p-median. Location Science, 5 (4), 207-226. (1997)
9. Hochbaum, D. S., Shmoys, D. B.: A best possible heuristic for the k-center problem. Mathematics of Operations Research, 10 (2), 180-184. (1985)
10. Ilhan, T., Pınar, M. C.: An efficient exact algorithm for the vertex p-center problem. Technical report, Department of Industrial Engineering, Bilkent University. (2001)
11. Kariv, O., Hakimi, S. L.: An algorithmic approach to network location problems. I: The p-Centers. SIAM Journal on Applied Mathematics, 37(3), 513-538. (1979)
12. López-Sánchez, A. D., Sánchez-Oro, J., Hernández-Díaz, A. G.: GRASP and VNS for solving the p-next center problem. Computers and Operations Research, 104, 295-303. (2019)
13. Minieka, E.: The m-center problem. Society for Industrial and Applied Mathematics, 12(1), 138–139. (1970)

14. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research, 24 (11), 1097–1100. (1997)
15. Mladenovic, N., Labbé, M., Hansen, P.: Solving the p-center problem with tabu search and variable neighborhood search. Networks 42 (1), 48–64. (2003)
16. Pullan, W.: A memetic genetic algorithm for the vertex p-center problem. Evol Comput, 16 (3), 417–436. (2008)
17. Ristic, D., Mladenovic, N., Todosijevic, R., Urosevic, D.: Filtered variable neighborhood search method for the p-next center problem. International Journal for Traffic and Transport Engineering, 11 (2), 294 - 309. (2021)

Dalibor Ristic graduated from Faculty of Electrical Engineering at Department of Computer Science, University of Nis, Serbia in 2007 and started his Ph.D. studies at the School of Computing, Union University, Belgrade, Serbia in 2013. From 2013 to 2017, he was a Teaching Assistant at the School of Computing, Union University where he gave lectures in Design and Analysis of Algorithms, Introduction to Programming, Object-Oriented Design and Methodology and Functional Programming. His main research areas are the development of complex graph algorithms and combinatorial optimization. Since 2008, he has worked as a software developer for a few software companies as a part of experienced teams that designed and developed complex systems in the fields of Energetics and Finance.

Dragan Urosevic is a research professor at the Mathematical Institute of the Serbian Academy of Sciences and Arts. He is a visiting professor at university in Novi Sad (Serbia). He received his B.Sc. degree at Faculty of Mathematics, University of Belgrade, in 1987. He received his M.Sc. degree at Faculty of Mathematics, University of Belgrade, in 1994, with the thesis “Heuristics for scheduling Parallel programs on Multiprocessor systems”. In 2004 he got PhD degree with the thesis “Solving problems on graphs by using Variable neighborhood search” at Faculty of Mathematics, University of Belgrade. Since 2001 he participated in a number of scientific projects, national, bilateral. He is a member of the Program Committees for several international conferences related to optimization and computer science fields. He also was the co-chair of the XIII Balkan Conference on Operational Research (BALCOR, Belgrade, Serbia, May 22-25, 2918). His main research interests include combinatorial optimization, mathematical programming, metaheuristics and computational complexity. He is a co-author of three chapters in monographs, more than 35 papers in refereed international journals, and more than 20 papers in international conference proceedings.

Nenad Mladenovic was born on April 28, 1951 in Jagodina, Serbia. He finished primary and secondary school in Belgrade. He graduated from the Department of Mathematics (major Cybernetics) at the Faculty of Natural Sciences and Mathematics, University of Belgrade in 1976. He received his master degree in 1982 from the Faculty of Organizational Sciences, University of Belgrade with a master's thesis titled “Comparative analysis of some nonlinear programming methods”. He defended his doctoral thesis “New nonlinear programming methods with application in location, allocation and transportation problems” in 1988 at the Faculty of Organizational Sciences. Dr Mladenović, as a visiting professor, visited several well-known world universities. From 2005-2013 he taught Operations Research, Heuristic Optimization and Operations Management at the Faculty of Mathematics, University of Birmingham and Brunel University in London. From 2013-2016, as an international chair, he gave lectures on Mathematical Optimization Methods to doctoral students at the University of Valenciennes in France. Dr Nenad Mladenović was a very productive researcher, who gained an enviable world reputation in the field of operations research, dealing with problems of decision-making and management in complex systems. He published about 200 papers in scientific journals, 70 papers in edited conference proceedings and 40 books and chapters in scientific monographs.

Raca Todosijevic is an Associated Professor in Computer Science at the Polytechnic University-Hauts-de-France, Valenciennes, France. His principle research line is combinatorial optimization with the special emphasis on an efficient and effective design of heuristic and matheuristic approaches. He has published more than 30 journal papers. He has been a member of program committees of several international conferences related to the domain of his research. In 2016, he received EURO Doctoral Dissertation Award 2016.

Received: August 04, 2021; Accepted: September 18, 2022.