Yugoslav Journal of Operations Research
34 (2024), Number 3, 475-497
DOI: https://doi.org/10.2298/YJOR230815039R

EXACT SOLUTIONS OF SOME OR-LIBRARY TEST INSTANCES FOR THE *P*-NEXT CENTER PROBLEM

Dalibor RISTIĆ

School of Computing, Union University, Belgrade, Serbia
dalibor.ristic@outlook.com

Raca TODOSIJEVIĆ

Polytechnic University of Hauts-de-France, Valenciennes, France
racatodosijevic@gmail.com

Dragan UROŠEVIĆ

Mathematical Institute of the Serbian Academy of Sciences and Arts, Belgrade, Serbia
School of Computing, Union University, Belgrade, Serbia
durosevic@raf.rs

Received: August 2023 / Accepted: February 2024

Abstract: OR-Library is a platform that provides standardized examples for testing problem-solving algorithms in many fields of the operational research and combinatorial optimization. One of the problems emerged in the previous decade is the *p*-next center problem. The solution to the *p*-next center problem implies locating *p* centers in order to minimize the maximum user distance to the closest center plus the distance between that center and the center closest to it. There are several heuristic algorithms for solving this NP-hard problem that give optimal or near-optimal solutions. We propose an algorithm for solving the *p*-next center problem based on the variable neighborhood search method, capable of recognizing whether some of the found solutions from the OR-Library test set are exact. As a result of the algorithm execution, more than 50% of the solutions are identified as globally optimal. The paper presents a table with the found exact solution values for the *p*-next center problem from the OR-Library test set.

Keywords: OR-Library test set, *p*-next center problem, variable neighborhood search, heuristic algorithms, combinatorial optimization.

MSC: 68T20, 90B06.

1. INTRODUCTION

In the 1980s, one of problems in the field of operational research was standardization of data sets for testing implemented algorithms. Comparing the efficiency of different algorithms requires testing them over the same test set. Apart from standardization, the problem was test set availability. As a solution to these problems, the OR-Library [1] distribution platform was introduced in 1990. The platform contains data sets for testing solutions to various operational research problems. Researchers could request and receive on their e-mail account any test set from the library.

The OR-Library contains data sets for testing solutions to the *p*-median problem, warehouse location, traveling salesman problem and many more. Over time, data sets from the OR-Library platform have been used to test solutions to various problems such as the *p*-center [2], *p*-median [3], *p*-next center [4-7] etc. Data sets are available at http://people.brunel.ac.uk/~mastjjb/jeb/info.html.

One of the more recent problems in the field of operational research is the *p*-next center problem (pNCP). The *p*-next center problem was presented in 2015 in [4] as an extension of the *p*-center problem [8]. It is defined as locating *p* out of *n* centers to minimize the maximum sum of the distance from user to the closest center and the distance between that center and its closest center. It is formally defined over an undirected weighted graph $G = (V, E)$, where V is the set of all nodes and E is the set of branches of the graph. The shortest distance between the nodes i and j is represented as $d(i, j)$.

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

Together with the definition of the pNCP, the paper [4] also presents several exact mathematical models that can be used to solve smaller problems. Also, the authors show in [4] that the pNCP is an NP-hard problem. The first heuristic algorithms for solving the *p*-next center problem were proposed by Lopez-Sanchez et al. in [5]. The authors present the GRASP and VNS implementations and the hybrid version as a combination of these two algorithms. The first algorithm, applied to larger instances of the *p*-next center problem, is the "*Filtered variable neighborhood search method for the p-next center problem*" [7] from 2021. Afterward, an improved implementation of the same VNS framework [6] was presented, which is currently best performing in solving the *p*-next center problem. All these methods and algorithms were tested on the OR-Library test set initially intended for the *p*-median problem. The test set contains 40 test instances with 100 to 900 nodes and *p* values between 5 and 200.

This paper provides exact solutions to larger test instances of the *p*-next center problem from the OR-Library test set. The solutions can be used to evaluate and compare different algorithms for solving the *p*-next center problem. To this end, in the next chapter, we first present the algorithm used to solve the pNCP. In the third chapter, we give a table of the obtained results of the algorithm executions on the OR-Library test set, i.e., the exact solutions to *p*-next center problems represented by the aforementioned test set. Finally, in Chapter 4, we conclude the paper with a brief summary and announcement of future work.

2. ALGORITHM

In this paper, we introduce an algorithm which might identify exact solutions to the *p*-next center problem. The algorithm is built on a variable neighborhoods search (VNS) metaheuristic. VNS is presented in the 1997 by Mladenović and Hansen [9] as a generic method for building search algorithms. Starting from a predefined current solution, the basis of VNS implementation is systematic change of the searched solution and a local search of the current solution with the aim of finding a local optimum. The incumbent solution is the best solution found so far. The change of the solution, i.e., the next current solution is selected as a random solution from the N_k neighborhood of the previously searched solution. The N_k neighborhood presents the set of solutions generated by replacing exactly k elements of the considered solution with new elements that were previously not contained in the solution. If the local search from so randomly selected solution finds no better solution than the currently incumbent solution, the k value increases by 1, but if after increasing becomes greater than predefined maximum value k_{max}, then resets to1. Otherwise, if the better solution is found, k is reset to 1. A detailed explanation of the basic variable neighborhood search method can be found in [9].

An efficient implementation of the VNS method for solving the *p*-next center problem is given in [6]. The algorithm from that paper, applied to the OR-Library test set, found so far the best-known solutions to the *p*-next center problem. However, it is a simple heuristic algorithm that cannot guarantee the optimality of its solution. In this paper, we present a modification of the algorithm from [6] capable of recognizing whether the found solutions to particular instances of the *p*-next center problem are globally optimal.

In the paper [6], it is explained that the algorithm iteratively improves the current solution, i.e., in each of the local search iterations it tries to find a better solution than the current one. It is noticed that the function value in the case of the *p*-next center problem corresponds to the maximum sum of the user's distance to the closest center and the distance between that center and the center closest to it. The user corresponding to the maximum distance is called the critical user. The center closest to a user is the reference center, and the center closest to it is the backup center. The essence of the local search phase [6] is replacing one of the centers of the current solution with a new one so that the function value corresponding to the critical user is decreased. The critical user needs to be allocated a new reference center, closer than the previous one, or a new backup center closer to the reference center. In this way, the function value for the critical user is reduced, but the improvement of the current solution and the solution optimality is not guaranteed. The algorithm we propose in this paper expands the solution [6] and guarantees the optimality of the new solution in the N_1 neighborhood of the current solution, though at the expense of the time complexity of the algorithm, and thus the time it takes to find a solution. To present the complete algorithm, we take Figure 1 and the example from [6], which we later expand with the explanation and illustration in Figure 2. So, let us first look at the example of the *p*-next center problem with $n = 15$ users and $p = 3$ centers as given in the following figures.

Based on Figure 1, the current solution to the *p*-next center problem with $n = 15$ users and $p = 3$ centers contains three reference center 2, 5 and 11. The critical user is node $u_c = 8$. To improve the current solution, we can look for a new reference center closer to critical user 8 (within circle C_1 if there is already a center within radius $d(8,5) + d(5,2) -$

$d(8, v)$ around the new center v, e.g., center 10) or a new backup center closer to the reference center 5 (inside circle C_2). On the other hand, if there is another center at the same distance as the reference center 5 (on circle C_1), we can look for a new backup center with a distance to that center closer than that of the respective backup center to the previous reference center.

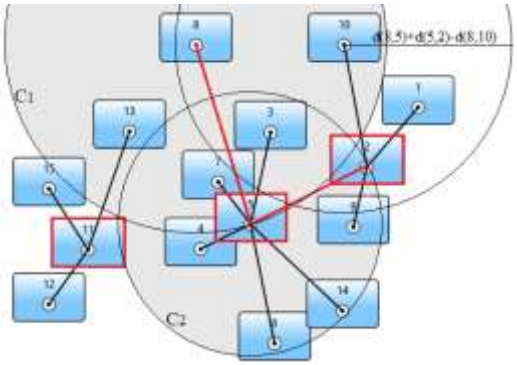

Figure 1: Example of the *p*-next center problem with n = 15 and p = 3; the current solution is P = {2, 5, 11} and the critical user u_c = 8

Let us denote the reference center of the critical user as the critical center p_c, and its backup center as p_c^*. If p_c, i.e., center 5, is closed, there are several other potential ways to decrease the function value of the current solution. Based on Figure 2, we can find a new reference or backup center closer to center 2 in the area A_1. As center 2, denoted by p', is the second closest center to the critical user and the radius of the area A_1 is $d(8,5) + d(5,2) - d(8,2)$, the new function value for the user u_c after opening the center in the area A_1 is certainly lower than the previous value $d(8,5) + d(5,2)$. On the other hand, if there is at least one other center in the area A_1 apart from center 2, it is enough to close the p_c, and after opening any center not closer to the critical user than center p', the current solution will potentially be improved. Similarly, any center p at the same distance from the critical user as the center p' (on the outer circle A_2) is a potentially new reference center if the newly-added (not closer to the critical user) or any already existing center is located within a radius of $d(8,5) + d(5,2) - d(8,p)$ around the center p. Also, it is necessary to consider all centers in the area A_2, since each of them is potentially a new reference center v. If there is at least one center within the radius of $d(u_c, p_c) + d(p_c, p_c^*) - d(u_c, v)$ around the new center v, v becomes the new reference center of the critical user and the function value decreases.

Considering the above, we find the center that needs to be added to the current solution to reduce the function value corresponding to the critical user. In other words, we filter out potentially new centers so that only those that reduce the value of the function corresponding to the critical user are considered. Although we do not reduce algorithm complexity in the worst case, we narrow the search scope and accelerate the convergence towards the local optimum by filtering potential solutions. However, opening a new center is not enough to make a new solution better than the current one. Whether it will be better

depends on other users who will lose some of their centers in the new solution. It may happen that some of the users, who are not a critical user in the previous solution, are allocated new centers so that the function value is greater than the previous value corresponding to the critical user. Therefore, it is necessary to optimally choose the center to be closed so that the new objective function value is as low as possible. The method that optimally identifies the center to be closed is explained in detail in [6]. The implementation is also given in Appendix 1, presenting complete pseudocode.

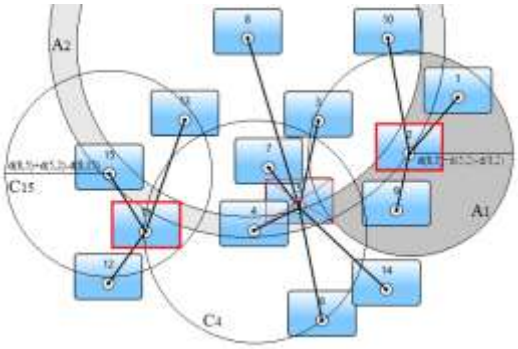

Figure 2: Example of the *p*-next center problem with $n = 15$ and $p = 3$; the current solution is $P = \{2, 5, 11\}$; the critical user $u_c = 8$ and the critical center $p_c = 5$ is going to be closed

This paper focuses on extending the algorithm from [6], which allows recognizing optimal solutions. To this end, we first present a property based on which we further implement this modification.

Property 1: Let u_c be the critical user, p_c its reference center, p_c' its second-closest center and $c1(v)$ the closest center to the user v ($v \in V$) in the current solution P. Then, if $d(u, v)$ represents the shortest distance between the u and v nodes and there is a better solution in the neighborhood $N_1(P)$ than the current solution P, at least one of the following propositions must be true:

(i) the new center c_{in} is not farther from the critical user than the center p_c ($d(c_{in}, u_c) \leq d(p_c, u_c)$) and $d(u_c, c_{in}) + d\big(c_{in}, c1(c_{in})\big) < d(u_c, p_c) + d\big(p_c, c1(p_c)\big)$ (circle C_1 in Figure 1),

(ii) the new center c_{in} is not closer to the critical user than the center p_c ($d(c_{in}, u_c) \geq d(p_c, u_c)$), there is a center p ($p \in P$, including p_c) such that it is at the same distance from the critical user as the center p_c ($d(p, u_c) = d(p_c, u_c)$) and $d(p, c_{in}) < d\big(p_c, c1(p_c)\big)$,

(iii) the center p_c is deleted, the new center c_{in} is not farther from the critical user than the center p_c' (area A_2 in Figure 2: $d(c_{in}, u_c) \leq d(p_c', u_c)$) and there is a center p ($p \in P$ and $p \neq p_c$) so that $d(u_c, c_{in}) + d(c_{in}, p) < d(u_c, p_c) + d\big(p_c, c1(p_c)\big)$,

(iv) the center p_c is deleted, the new center c_{in} is not closer to the critical user than the center p_c' ($d(c_{in}, u_c) \geq d(p_c', u_c)$) and there are centers p_r and p_b ($p_r \in P$, $p_b \in P \cup \{c_{in}\}$ and $p_r \neq p_b \neq p_c$) so that $d(p_r, u_c) = d(p_c', u_c)$ and $d(u_c, p_r) + d(p_r, p_b) < d(u_c, p_c) + d\big(p_c, c1(p_c)\big)$ (such as area A_1 in Figure 2).

Proof. The objective function value is determined by the sum of the distances to the centers allocated to the critical user: $f(P) = d(u_c, p_c) + d\big(p_c, c1(p_c)\big)$. So that the new solution from the neighborhood $N_1(P)$ would be better than the current solution P, a new reference or/and a backup center must be allocated to the critical user u_c. If the closest center p_c remains open in a new solution, one of these centers must be the new center c_{in}. Let us consider cases when the center p_c is deleted and when it is not deleted:

1) Reference center p_c is not deleted. Suppose none of the propositions (i) and (ii) are true. The new center c_{in} can be found inside the circle C_1, on the circular arc C_1 and outside the circle C_1 (see Figure 1).

- If the new center is inside the circle, it will be the closest to the critical user u_c, i.e., the new reference center of the user u_c. Since (i) is not true, it holds that $d(u_c, c_{in}) + d\big(c_{in}, c1(c_{in})\big) \geq f(P)$, which is in contradiction with the fact that the new solution is better than the current solution P.
- In case that c_{in} is on the circular arc C_1, the user u_c can keep the same reference center p_c or get a new one, either c_{in} or any of centers $p \in P$ that are at the same distance from u_c as the center p_c. Since (i) and (ii) are not true, it holds that $d(u_c, c_{in}) + d\big(c_{in}, c1(c_{in})\big) \geq f(P)$, as well as for all other centers p located on the circular arc C_1 ($d(u_c, p) = d(u_c, p_c)$), including p_c, that $d(u_c, p) + d(p, c_{in}) \geq f(P)$. Therefore, by allocating the center c_{in} to the critical user, either as a new reference or backup center, the objective function value is not improved.
- When the new center c_{in} is outside the circle C_1, the reference center of the critical user can remain p_c or become one of the centers $p \in P$ that are at the same distance from u_c as the center p_c. Since (ii) is not true, for each of the centers p located on the circular arc C_1 (including p_c), it holds that $d(u_c, p) + d(p, c_{in}) \geq f(P)$. Therefore, by allocating the new backup center c_{in} to the critical user, the objective function value is not improved.

2) Reference center p_c is deleted. Let us assume that propositions (iii) and (iv) are not true. The new center c_{in} can be found inside the area A_2, on the outer circular arc A_2 and outside this area (see Figure 2).

- If the center c_{in} is placed inside the inner circular arc A_2, the correctness of the property can be shown similarly as in the first case from 1).
- Anyway, if the new center c_{in} is within the area A_2, it will be the closest to the critical user u_c, i.e., the new reference center of the user u_c. Let the center p be the new backup of the center c_{in}, i.e., $p = c1(c_{in})$ providing $c1(c_{in}) \neq p_c$, and otherwise p is the second-closest center of the user c_{in} in the solution P. Since (iii) is not true, it holds that $d(u_c, c_{in}) + d(c_{in}, p) \geq f(P)$, which is in contradiction with the fact that the new solution is better than the current solution P.
- In case that c_{in} is on the outer circular arc A_2, the user u_c as a new reference center gets one of the centers $p \in P \cup \{c_{in}\}$ that are located at the same distance from u_c as the center p_c', including p_c' and c_{in}. Let the center p' be the backup of the center p in the solution $P \cup \{c_{in}\} \backslash \{p_c\}$. Since (iii) and (iv) are not true, for each center p (including c_{in}), it holds that $d(u_c, p) + d(p, p') \geq f(P)$, which is again in contradiction with the fact that a better solution is found.
- When the new center c_{in} is outside the outer arc of the area A_2, the user u_c as a new reference center gets one of the centers $p \in P$ that are at the same distance from u_c as

the center p'_c, including p'_c. Let the center p' be the backup of the center p in the solution $P \cup \{c_{in}\} \backslash \{p_c\}$. Since (iv) is not true, for each center p, it holds that $d(u_c, p) + d(p, p') \geq f(P)$. Therefore, the objective function value is not improved.

The reference center p_c can remain open or be closed in a new solution from the $N_1(P)$ neighborhood. The new center c_{in} can be found inside the circle C_1, outside the circle or on the circular arc C_1, and also on the outer circular arc A_2, inside the area or outside the area A_2. Therefore, according to 1) and 2), it follows that the property is correct.

The following property enables us to recognize the exact solutions to particular instances of the *p*-next center problem.

Property 2: Let the critical user u_c be also the center included in the current solution P, and $c1(u_c)$ its closest center in the solution P, ($u_c \in P$, $c1(u_c) \in P$ and $c1(u_c) \neq u_c$). Then, if in the set of potentially new centers there is not any center c_{in} ($c_{in} \notin P$) that is closer to the critical user u_c than the closest center $c1(u_c)$, the current solution P is the optimal solution for the *p*-next center problem.

Proof. Let $f(P)$ be the objective function value in the current solution P, and $f(u_i)$ the function value for the user u_i. In addition, $d(u_i, u_j)$ represents the shortest distance between the u_i and u_j nodes. Then, it holds that:

$$f(P) = \max_{i=1,\ldots,n} f(u_i) = f(u_c) = d(u_c, u_c) + d\big(u_c, c1(u_c)\big) = d\big(u_c, c1(u_c)\big). \quad (2)$$

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 at least one new center c_{in} into the current solution, so that:

$$d(u_c, c_{in}) < f(P), \quad (3)$$

i.e.,

$$d(u_c, c_{in}) < d\big(u_c, c1(u_c)\big) \quad (4)$$

If it is not possible to find a new center c_{in} that satisfies inequality (4), it means that it is not possible to improve the current solution, i.e., the current solution is the optimal solution to the *p*-next center problem. ■

Recall that in the local search phase of the VNS implementation, we try to reduce the objective function value by replacing one of the current solution (P) centers with a new center c_{in} ($c_{in} \notin P$). We narrow the search scope only to centers that potentially improve the current solution, i.e., reduce the function value for the critical user u_c. Based on Property 1, we implement a procedure (Algorithm 1) that certainly checks whether it is possible to improve the current solution by opening a new center.

The algorithm assumes that the current solution P represents the first p elements of the array x_{cur}, and the other users are $n - p$ elements from the end of the same array, where n is the number of users. Additionally, the presented algorithm uses the following structures:

- *dist(u, v)* - the shortest distance between the nodes *u* and *v*,
- *c1(u)* - the closest center to the user *u* in the current solution,
- *c2(u)* - the second-closest center to the user *u* in the current solution.

Algorithm 1: Checking whether there is a better solution if c_{in} is opened as a new center

```
ExistsRelaxedDistance(x_cur, c1, c2, u_c, c_in)
p_c = c1(u_c)
f = dist(u_c, p_c) + dist(p_c, c1(p_c))

**Property 1(i):
If dist(c_in, u_c) ≤ dist(p_c, u_c) and dist(u_c, c_in) + dist(c_in, c1(c_in)) < f
     Return True
End If

**Property 1(ii):
If dist(p_c, c_in) < dist(p_c, c1(p_c))
     Return True
End If

**Property 1(ii):
If dist(c2(u_c), u_c) = dist(p_c, u_c)
     For Each p_r ∈ {x_cur(1), …, x_cur(p)} \ {p_c}
        If dist(u_c, p_c) = dist(u_c, p_r) and dist(p_r, c_in) < dist(p_c, c1(p_c))
             Return True
        End If
     End For Each
End If

**Property 1(iii) – p_c is deleted:
If dist(c_in, u_c) ≤ dist(c2(u_c), u_c)
     p = c1(c_in) if c1(c_in) ≠ p_c else c2(c_in)
     If dist(u_c, c_in) + dist(c_in, p) < f
             Return True
     End If
End If

**Property 1(iv) – p_c is deleted:
If dist(c2(u_c), u_c) ≤ dist(c_in, u_c)
     For Each p_r ∈ {x_cur(1), …, x_cur(p)} \ {p_c}
        If dist(u_c, p_r) = dist(u_c, c2(u_c))
             If dist(u_c, p_r) + dist(p_r, c_in) < f
                     Return True
             End If
             p_b = c1(p_r) if c1(p_r) ≠ p_c else c2(p_r)
             If dist(u_c, p_r) + dist(p_r, p_b) < f
                     Return True
             End If
        End If
     End For Each
End If
Return False.
```

Based on Property 2, we propose a modification of the VNS algorithm from [6], capable of recognizing in some cases exact solutions to the *p*-next center problem. The idea is to check whether the critical user is one of the current solution centers, and if it is, to proceed with the local search phase only if there is at least one center that potentially improves the current solution. If not, the execution stops and the algorithm identifies the current solution as the global optimum (Algorithm 2). The algorithm execution is time-limited by the value of parameter t_{max}.

Algorithm 2: Shaking procedure for the *p*-next center problem

```
VariableNeighborhoodSearch(k_max, t_max)
**Initialization: Randomly initialize x_opt;
according to x_opt initialize arrays c1 and c2, f_opt, u_c;
copy initial solution into the current one, i.e., copy f_opt, x_opt, c1, c2
and u_c into f_cur, x_cur, c1', c2' and u_c', respectively.

Repeat the Main step until the stopping condition is met (time ≤ t_max)
Main step:
     k ← 1
     While k ≤ k_max

        If u_c' is center
             c_in ← find center from {x_cur(p + 1), …, x_cur(n)}
                    where d(u_c', center) < d(u_c', c1'(u_c'))
             If c_in is not found
                    **the optimal solution has been found**
                    Return {x_opt(1), …, x_opt(p)}, f_opt
             End If
        End If

        Shaking operator:
        **generate a solution at random from kth neighborhood**
        For Each j = 1, ..., k
             • Take center to be inserted (c_in)
               if ExistsRelaxedDistance(x_cur, c1', c2', u_c', x_cur(c_in))
             • Find center to be deleted (c_out) at random
             • Update x_cur, c1' and c2', i.e., execute:
               x_cur(c_in) ↔ x_cur(c_out)
               Update(x_cur, c1', c2')
             • Update f_cur and u_c' according to x_cur, c1' and c2'
        End For Each

        Local search:
        If any potentially better solution found
             x_cur, c1', c2', u_c', f_cur ← LocalSearchVertexSubstitution(x_cur,
                                        c1', c2', u_c', f_cur)
             Move or not:
             If f_cur ≤ f_opt
                  **save current solution as the optimal; return to N_1**
                  x_opt ← x_cur; f_opt ← f_cur; u_c ← u_c'; c1 ← c1'; c2 ← c2'
                  k ← 1
             Else
                  **reject the new solution; change the neighborhood**
                  x_cur ← x_opt; f_cur ← f_opt; u_c' ← u_c; c1' ← c1; c2' ← c2
                  k ← k + 1
             End If
        End If

     End While
Return {x_opt(1), …, x_opt(p)}, f_opt.
```

In the local search phase, in order to accelerate the convergence towards the local optimum, potential centers are filtered out based on the call of the *Exists Relaxed Distance* procedure. It is obvious that the complexity of the *Exists Relaxed Distance* affects the local search execution time, as presented in the following properties.

Property 3: The time complexity of the *Move* algorithm (Algorithm 3 in Appendix 1) is $O(n+p^2)$.

Proof. Let:

- $P=\{p_1,p_2,\dots,p_p\}$ – be the current solution to the p-next center problem,
- $V=\{v_1,v_2,\dots,v_n\}$ – be the set of all users,
- $P'=P\cup\{c_{in}\}$, $c_{in}\in V\backslash P$ – be the solution that is obtained by adding a center c_{in} to the current solution,
- $P_i=P'\backslash\{p_i\}=P\cup\{c_{in}\}\backslash\{p_i\}$, $p_i\in P$ – be the solution that is obtained by deleting the center p_i from the solution P',
- $r(p_i,p_j)$, $p_i\in P'$, $p_j\in P'$ and $p_i\neq p_j$ – be the maximum function value among all users to whom centers (p_i,p_j) are allocated as the reference and backup center in the solution P',
- $z(p_i)$, $p_i\in P$ – be the maximum objective function value (after closing center p_i) taking into account all users to whom the center p_i was allocated either as a reference or backup center and deleted from the solution P',
- $rc_X(v_j)$, $j=1,\dots,n$ – be the reference center for the user v_j in the solution $X\in\{P',P_i\}$,
- $c1_X(v_j)$, $j=1,\dots,n$ – be the closest center to the user/center v_j in the solution $X\in\{P,P',P_i\}$,
- $c2_X(v_j)$, $j=1,\dots,n$ – be the second closest center to the user/center v_j in the solution $X\in\{P,P'\}$,
- $d(u,v)$ – be the shortest distance between the u and v nodes.

Let us first consider the set of centers P', i.e., the case when the new center c_{in} is included in the current solution P, without any center being closed. For each $v_i\in V$, in the solution P' applies:

$$c1_{P'}(v_i)=\begin{cases} c_{in},\ if\ cndCP_1(v_i)\vee\big(cndCP_2(v_i)\wedge cndCP_3(v_i)\big)\\ c1_P(v_i),\ if\ cndCP_4(v_i)\vee\big(cndCP_5(v_i)\wedge cndCP_6(v_i)\big),\\ c2_P(v_i),\ otherwise\end{cases} \tag{5}$$

where:

$$cndCP_1(v_i)=d(v_i,c_{in})<d\big(v_i,c1_P(v_i)\big),$$

$$cndCP_2(v_i)=$$

$$\big[d(v_i,c_{in})=d\big(v_i,c1_P(v_i)\big)\big]\wedge d\big(c_{in},c1_P(c_{in})\big)<$$
$$min\Big\{d\Big(c1_P(v_i),c1_P\big(c1_P(v_i)\big)\Big),d(c1_P(v_i),c_{in})\Big\},$$

$$cndCP_3(v_i)=$$

$$d(v_i,c_{in})<d\big(v_i,c2_P(v_i)\big)\vee d\big(c_{in},c1_P(c_{in})\big)<$$
$$min\Big\{d\Big(c2_P(v_i),c1_P\big(c2_P(v_i)\big)\Big),(c2_P(v_i),c_{in})\Big\},$$

$$cndCP_4(v_i)=d\big(v_i,c1_P(v_i)\big)<d\big(v_i,c2_P(v_i)\big),$$

$$cndCP_5(v_i) = \left[d\left(v_i, c1_P(v_i)\right) = d\left(v_i, c2_P(v_i)\right)\right],$$

$$cndCP_6(v_i) =$$

$$min\left\{d\left(c1_P(v_i), c1_P\left(c1_P(v_i)\right)\right), d(c1_P(v_i), c_{in})\right\} \leq$$

$$min\left\{d\left(c2_P(v_i), c1_P\left(c2_P(v_i)\right)\right), (c2_P(v_i), c_{in})\right\}.$$

Then:

$$rc_{P'}(v_i) = \begin{cases} c1_{P'}(v_i), v_i \notin P' \\ v_i, v_i \in P' \end{cases} \quad (6)$$

Let us assume that:

$c1_{P'}(v_i) = c_{in}$, then:

$$c2'_{P'}(v_i) = \begin{cases} c1_P(v_i), \ cndCP'_1(v_i) \vee \left(cndCP'_2(v_i) \wedge cndCP'_3(v_i)\right) \\ c2_P(v_i), \ otherwise \end{cases}, \quad (7)$$

where:

$$cndCP'_1(v_i) = d\left(v_i, c1_P(v_i)\right) < d\left(v_i, c2_P(v_i)\right),$$

$$cndCP'_2(v_i) = \left[d\left(v_i, c1_P(v_i)\right) = d\left(v_i, c2_P(v_i)\right)\right],$$

$$cndCP'_3(v_i) =$$

$$min\left\{d\left(c1_P(v_i), c1_P\left(c1_P(v_i)\right)\right), d(c1_P(v_i), c_{in})\right\} \leq$$

$$min\left\{d\left(c2_P(v_i), c1_P\left(c2_P(v_i)\right)\right), (c2_P(v_i), c_{in})\right\}.$$

1) $c1_{P'}(v_i) = c1_P(v_i)$:

$$c2''_{P'}(v_i) = \begin{cases} c_{in}, \ cndCP''_1(v_i) \vee cndCP''_2(v_i) \\ c2_P(v_i), \ otherwise \end{cases}, \quad (8)$$

where:

$$cndCP''_1(v_i) = d(v_i, c_{in}) < d\left(v_i, c2_P(v_i)\right),$$

$$cndCP''_2(v_i) =$$

$$\left[d(v_i, c_{in}) = d\left(v_i, c2_P(v_i)\right)\right] \wedge d\left(c_{in}, c1_P(c_{in})\right) <$$

$$min\left\{d\left(c2_P(v_i), c1_P\left(c2_P(v_i)\right)\right), (c2_P(v_i), c_{in})\right\}.$$

2) $c1_{P'}(v_i) = c2_P(v_i)$:

$$c2'''_{P'}(v_i) = \begin{cases} c_{in}, \ cndCP'''_1(v_i) \wedge cndCP'''_2(v_i) \\ c1_P(v_i), \ otherwise \end{cases}, \quad (9)$$

where:

$$cndCP'''_1(v_i) = \left[d(v_i, c_{in}) = d\left(v_i, c1_P(v_i)\right)\right],$$

$$cndCP_2'''(v_i) = d\big(c_{in}, c1_P(c_{in})\big) < \\ min\Big\{d\Big(c1_P(v_i), c1_P\big(c1_P(v_i)\big)\Big), (c1_P(v_i), c_{in})\Big\}.$$

From (7), (8) and (9), it follows that:

$$c2_{P'}(v_i) = \begin{cases} c2'_{P'}(v_i),\ c1_{P'}(v_i) = c_{in} \\ c2''_{P'}(v_i),\ c1_{P'}(v_i) = c1_P(v_i) \\ c2'''_{P'}(v_i),\ c1_{P'}(v_i) = c2_P(v_i) \end{cases} \tag{10}$$

Based on the previous expressions, it follows that:

$$r\left(p_j, c1_{P'}(p_j)\right) = \max_{v_i \in V}\big\{d\big(v_i, p_j\big) \mid p_j = rc_{P'}(v_i)\big\} + d\left(p_j, c1_{P'}(p_j)\right) \tag{11}$$

Let us now consider the set P_i, i.e., the solution obtained when the center p_i is excluded from the set P'. Comparing the solutions P' and P_i, the set of users V is divided into two disjoint subsets:

- V_i', users that in the solution P_i keep their pair of centers from the solution P':

$$z'(p_i) = 0, \tag{12}$$

- V_i'', users that lose their reference and/or backup center from the solution P' after closing the center p_i:

$$z''(p_i) = \max_{v_j \in V_i''}\left\{d\left(v_j, rc_{P_i}(v_i)\right) + d\left(rc_{P_i}(v_i), c1_{P_i}\left(rc_{P_i}(v_i)\right)\right)\right\} \tag{13}$$

From (12) and (13), it follows:

$$z(p_i) = max\{z'(p_i), z''(p_i)\} = z''(p_i) = \max_{v_j \in V_i''}\left\{d\left(v_j, rc_{P_i}(v_i)\right) + \right. \\ \left. d\left(rc_{P_i}(v_i), c1_{P_i}\left(rc_{P_i}(v_i)\right)\right)\right\} \tag{14}$$

Also, for each $v_i \in V$, in the solution P_i, it holds that:

$$c1_{P_i}(v_i) = \\ \begin{cases} c1_{P'}(v_i),\ cndCP_1^{iv}(v_i) \vee \Big(cndCP_2^{iv}(v_i) \wedge \Big(cndCP_3^{iv}(v_i) \vee cndCP_4^{iv}(v_i)\Big)\Big), \\ c2_{P'}(v_i),\ otherwise \end{cases} \tag{15}$$

where:

$$cndCP_1^{iv}(v_i) = [c2_{P'}(v_i) = p_i],$$

$$cndCP_2^{iv}(v_i) = c1_{P'}(v_i) \neq p_i,$$

$$cndCP_3^{iv}(v_i) = d\big(v_i, c1_{P'}(v_i)\big) < d\big(v_i, c2_{P'}(v_i)\big),$$

$$cndCP_4^{iv}(v_i) = d\left(c1_{P'}(v_i), bc_{P_i}\big(c1_{P'}(v_i)\big)\right) < d\left(c2_{P'}(v_i), bc_{P_i}\big(c2_{P'}(v_i)\big)\right),$$

$$bc_{P_i}(c) = \begin{cases} c1_{P'}(c), c1_{P'}(c) \neq p_i \\ c2_{P'}(c), otherwise \end{cases},\ c \in \{c1_{P'}(v_i), c2_{P'}(v_i)\}.$$

Then:

$$rc_{P_i}(v_i) = \begin{cases} c1_{P_i}(v_i), v_i \notin P_i \\ v_i, v_i \in P_i \end{cases} \tag{16}$$

Finally, based on the previous expressions, we present *Best deletion* as follows:

$$\begin{aligned} f\left(P^{(best)}\right) &= \min_{i=1,\dots,p} f(P_i) \\ &= \min_{i=1,\dots,p} \left\{ \max_{v_j \in V} \left\{ d\left(v_j, rc_{P_i}(v_j)\right) + d\left(rc_{P_i}(v_j), c1_{P_i}\left(rc_{P_i}(v_j)\right)\right)\right\}\right\} \\ &= \min_{i=1,\dots,p} \left\{ \max_{v_j \in (V_i' \cup V_i'')} \left\{ d\left(v_j, rc_{P_i}(v_j)\right) + d\left(rc_{P_i}(v_j), c1_{P_i}\left(rc_{P_i}(v_j)\right)\right)\right\}\right\} \\ &= \min_{i=1,\dots,p} \left\{ max \left\{ \begin{matrix} \max_{v_j \in V_i'} \left\{ d\left(v_j, rc_{P_i}(v_i)\right) + d\left(rc_{P_i}(v_i), c1_{P_i}\left(rc_{P_i}(v_i)\right)\right)\right\}, \\ \max_{v_j \in V_i''} \left\{ d\left(v_j, rc_{P_i}(v_i)\right) + d\left(rc_{P_i}(v_i), c1_{P_i}\left(rc_{P_i}(v_i)\right)\right)\right\} \end{matrix} \right\}\right\} \\ &= \min_{i=1,\dots,p} \left\{ max \left\{ \max_{\substack{p_j \in P \cup \{c_{in}\} \setminus \{p_i\} \\ c1_{P'}(p_j) \neq p_i}} \left\{ r\left(p_j, c1_{P'}(p_j)\right)\right\}, z(p_i) \right\}\right\} \\ &= \min_{p_i \in P} \left\{ max \left\{ z(p_i), \max_{\substack{p_j \in P \cup \{c_{in}\} \setminus \{p_i\} \\ c1_{P'}(p_j) \neq p_i}} \left\{ r\left(p_j, c1_{P'}(p_j)\right)\right\} \right\}\right\}. \end{aligned} \tag{17}$$

The last expression leads to the procedure given in the *Best deletion* step in Algorithm 3.5 (Appendix 1). It follows from (5), (6), (15) and (16) that $c1_X(v_i)$ and $rc_X(v_i)$, where $X \in \{P', P_i\}$, are found in $O(1)$ time. Based on (11) and (14), it is simple to notice that *r* and *z* values can be found in $O(n)$ time. From (17), the *Best deletion* step is executed in $O(p^2)$, so the time complexity of the *Move* procedure is $O(n + p^2)$. ■

Property 4: The time complexity of a single iteration of the *Local Search Vertex Substitution* algorithm (Algorithm 5 in Appendix 1) is $O(n^2 + p^2 n)$ in the worst case.

Proof. In order to determine the time complexity of an iteration of the *Local Search Vertex Substitution* algorithm, it is first necessary to determine the complexity of the other procedures. According to Property 3, the time complexity of the *Move* procedure is $O(n + p^2)$. The *Update* procedure (Algorithm 4 in Appendix 1) updates the elements of arrays *c1* and *c2* for each of the *n* users and potential *p* reference centers, so that the time complexity is $O(pn)$. The time complexity of the *Exists Relaxed Distance* procedure is $O(p)$. The *Local Search Vertex Substitution* algorithm in the worst case uses *Exists Relaxed Distance* and *Move* procedures *n* - *p* times and *Update* procedure once, so the worst-case time complexity is $O(n^2 + p^2 n) + O(pn) + O(n) \approx O(n^2 + p^2 n)$. ■

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 32 GB RAM. It was performed 20 times on each test instance from the OR-Library test set, always starting with a different initial solution. Different combinations of parameters $k_{max} = p/2$, $k_{max} = p$ and $t_{max} = n$ and $t_{max} = 2n$ were tested. It turned out that the results were slightly better with higher values of the k_{max} parameter. On the other hand, the algorithm found the best solution well before the execution time limit expired. Therefore, we decided to present the results only for $k_{max} = p$ and $t_{max} = n$ seconds. The results are presented in the following tables.

Table 1 shows the results obtained for each instance from the OR-Library set. The first column of the table contains the name of the instance. The next three columns give the value p (number of centers), n (number of users) and "*Best Know Value*", i.e., the best solution known in the literature. In the "*Best Found Value*" column, we show the value of the best solution the algorithm found in 20 executions. The "*Time*" (or time-to-target) column shows the average time in seconds it took to find the best solution for the first time. "*Time*" is not the total execution time. The algorithm is executed until the time limit expires, i.e., for n seconds, or until it identifies a globally optimal solution. The "*#Best Known*" column gives the number of times the algorithm found the best known solution in 20 executions. We also give the percentage deviation of the best found solution compared to the best known solution in the "*Gap*" column. As the best known solution, we take the best solution found by the algorithm from [6]. The percentage gap is calculated as $\frac{Best\ found - Best\ known}{Best\ known} * 100$. Finally, the last column "*Exact Value*" shows whether the exact solution has been found.

Table 1: Results of the algorithm for OR-Library test instances

	p	n	Best Known Value	Best Found Value	Time	#Best Known	Gap (best found-best known) /best known	Exact Value
pmed1	5	100	166	166	0.10	20	0.00	
pmed2	10	100	135	135	1.34	20	0.00	
pmed3	10	100	151	151	3.00	20	0.00	
pmed4	20	100	118	118	5.85	20	0.00	
pmed5	33	100	85	85	0.76	16	0.00	✓
pmed6	5	200	107	107	6.60	20	0.00	
pmed7	10	200	84	84	35.75	15	0.00	
pmed8	20	200	81	84	7.73	0	3.70	
pmed9	40	200	71	71	12.46	16	0.00	✓

pmed10	67	200	70	70	0.79	20	0.00	✓
pmed11	5	300	70	70	2.26	19	0.00	
pmed12	10	300	72	72	2.49	14	0.00	✓
pmed13	30	300	47	47	59.69	1	0.00	
pmed14	60	300	60	60	13.36	20	0.00	✓
pmed15	100	300	44	44	68.38	14	0.00	✓
pmed16	5	400	54	54	86.95	19	0.00	
pmed17	10	400	46	47	23.68	0	2.17	
pmed18	40	400	50	50	26.76	17	0.00	✓
pmed19	80	400	32	37	250.07	0	15.63	
pmed20	133	400	40	40	222.64	13	0.00	✓
pmed21	5	500	48	48	51.18	15	0.00	
pmed22	10	500	49	49	22.45	3	0.00	
pmed23	50	500	31	37	209.20	0	19.35	
pmed24	100	500	33	33	317.23	2	0.00	✓
pmed25	167	500	44	44	29.03	20	0.00	✓
pmed26	5	600	47	47	55.67	16	0.00	
pmed27	10	600	38	38	210.91	2	0.00	
pmed28	60	600	57	57	2.79	20	0.00	✓
pmed29	120	600	36	36	158.65	20	0.00	✓
pmed30	200	600	40	40	32.45	20	0.00	✓
pmed31	5	700	35	35	30.10	16	0.00	
pmed32	10	700	72	72	4.17	20	0.00	✓
pmed33	70	700	22	28	406.79	0	27.27	
pmed34	140	700	41	41	12.86	20	0.00	✓
pmed35	5	800	36	36	24.75	5	0.00	
pmed36	10	800	42	42	12.77	20	0.00	✓

pmed37	80	800	33	33	176.91	20	0.00	✓
pmed38	5	900	40	40	13.81	18	0.00	✓
pmed39	10	900	74	74	9.06	20	0.00	✓
pmed40	90	900	23	25	517.58	0	8.70	
				Average:	**78.23s**	**13.53**	**1.92%**	**Total: 19**

Table 1 shows that the proposed algorithm is not as efficient as algorithm [6]. It found the best known solution in 13.53 out of 20 executions on average. The algorithm from [6] is 1.92% more successful. On the other hand, the algorithm was able to identify optimal solutions for 19 of the 40 instances, as indicated in the last column of the table. In any case, there are 6 instances for which the best known solution has not been found. Therefore, we decided to execute the algorithm once more time but now with the best known solutions as the initial ones. The obtained results are presented in the following table. The table is expanded with a column containing the number of branches (m) of the graph which represents the test instance.

Table 2: Exact results for OR-Library test instances

	p	n	m	Time	Exact Value
pmed5	33	100	200	0.01	85
pmed9	40	200	800	0.04	71
pmed10	67	200	800	0.04	70
pmed12	10	300	1800	0.23	72
pmed14	60	300	1800	0.25	60
pmed15	100	300	1800	0.14	44
pmed18	40	400	3200	0.08	50
pmed19	80	400	3200	0.29	32
pmed20	133	400	3200	0.52	40
pmed24	100	500	5000	1.02	33
pmed25	167	500	5000	0.65	44
pmed28	60	600	7200	1.95	57
pmed29	120	600	7200	1.98	36

pmed30	200	600	7200	1.93	40
pmed32	10	700	9800	3.18	72
pmed33	70	700	9800	3.25	22
pmed34	140	700	9800	3.23	41
pmed36	10	800	12800	4.80	42
pmed37	80	800	12800	1.10	33
pmed38	5	900	16200	7.20	40
pmed39	10	900	16200	7.02	74
pmed40	90	900	16200	0.99	23
Average:	**73.86**	**536.36**	**6909.01**	**1.81s**	**Total: 22**

Table 2 shows the results of the algorithm execution only for test instances that have been exactly solved. The algorithm was executed once for each OR-Library test instance with the same parameter values ($k_{max} = p$ and $t_{max} = n$ seconds) and the best known solutions as the initial solutions. The table shows that the algorithm was able to identify optimal solutions for 22 out of 40 instances. It is important to note that mostly larger instances are exactly solved. The average *p*, *n* and *m* values of exactly solved instances are 73.86, 536.36 and 6909.01, respectively.

4. CONCLUSION

The paper deals with identifying exact solutions of the *p*-next center problem instances presented by OR-Library [1] data set, initially intended for testing solutions to the *p*-median problem. The OR-Library platform for test set distribution has proven to be applicable and generally accepted among researchers for testing and comparing the efficiency of algorithms in various fields of the operational research and combinatorial optimization. After the *p*-next center [4] was introduced in 2015 as a new NP-hard problem, all so far proposed methods and algorithms which solve this problem have been tested on the "*p*-median" OR-Library test set. Therefore, it is vital to offer exact solutions for this data set, and especially for larger instances of problems that cannot be solved by exact mathematical models.

The solution to the *p*-next center problem is to locate *p* centers in order to minimize the maximum distance among *n* users to the closest center plus the distance between that center and the center closest to it. In the paper, we use a heuristic algorithm based on the variable neighborhood search method to solve the problem. The existing algorithm for solving the *p*-next center problem from [6] has been modified so that it is able to recognize some of the globally optimal solutions. Executed over the OR-Library test set, the algorithm turned out to be less efficient than the original algorithm [6], but was able to identify optimal

solutions for 22 out of 40 instances. These solutions are mostly examples of larger instances of the problem.

The proposed algorithm for the *p*-next center problem showed 55% success in identifying the exact solutions of the OR-Library test set. However, it would be a challenge to try to increase the success rate in future works, either by reducing time complexity as in algorithm [6] and/or tracking search history and, based on previous solutions, reducing the cardinality of a set of new potentially better solutions.

Funding. The research has been partially supported by the Serbian Ministry of Science, Innovations, and Technological Development, Agreement No. 451-03-66/2024-03/200029.

REFERENCES

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

[2] N. Mladenovic, M. Labbé and P. Hansen, "Solving the p-center problem with tabu search and variable neighborhood search," *Networks*, vol. 42(1), pp. 48–64, 2003.

[3] P. Hansen and N. Mladenovic, "Variable neighborhood search for the p-median," *Location Science*, vol. 5(4), pp. 207-226, 1997.

[4] M. Albareda-Sambola, Y. Hinojosa, A. Marín and J. Puerto, "When centers can fail: a close second opportunity," *Computers and Operations Research*, vol. 62, pp. 145–156, 2015.

[5] A. D. López-Sánchez, J. Sánchez-Oro and A. G. Hernández-Díaz, "GRASP and VNS for solving the p-next center problem," *Computers and Operations Research*, vol. 104, pp. 295-303, 2019.

[6] D. Ristic, N. Mladenovic, M. Ratli, R. Todosijevic and D. Urosevic, "Auxiliary data structures and techniques to speed up solving of the p-next center problem: A VNS heuristic," *Applied Soft Computing*, 2023.

[7] D. Ristic, N. Mladenovic, R. Todosijevic and D. Urosevic, "Filtered variable neighborhood search method for the p-next center problem," *International Journal for Traffic and Transport Engineering*, vol. 11(2), pp. 294 – 309, 2021.

[8] S. L. Hakimi, "Optimum distribution of switching centers in a communication network and some related graph theoretic problems," *Operations Research*, vol. 13(3), pp. 462-475, 1965.

[9] N. Mladenovic and P. Hansen, "Variable neighborhood search," *Computers and Operations Research*, vol. 24(11), pp. 1097–1100, 1997.

APPENDIX 1

Appendix 1 presents the pseudocode of the algorithm for solving the *p*-next center problem in its entirety. The code is taken from [6] with modification of *Exists Relaxed Distance* and *Variable Neighborhood Search* procedures to potentially identify the obtained solution as the globally optimal solution to the problem.

The algorithm in order to efficiently calculate the objective function value uses the arrays of the closest and second-closest centers: *c1* and *c2*. Also, the following auxiliary structures are used to find the optimal center for deletion during the local search phase:

- $r(p_i, p_j)$ - the maximum value of the pNCP function in the current solution, taking into account all users with p_i as a reference and p_j as a backup center and vice versa,

- $z(p_i)$ - the maximum value of the pNCP function among all users to whom center p_i was allocated as a reference or backup center in the current solution, but after center p_i was deleted from the solution.

The solution is represented by an array x_{cur} so that the first p elements of the array contain the current solution. The remaining $n - p$ elements from the end of the array, where n is the number of users, represent only the users in the current solution. Algorithm 3 describes the *Move* procedure, which based on the input parameters x_{cur}, c_{in}, *c1* and *c2* creates the *r* and *z* structures and returns a new function value f_{cur} and center c_{out}, which should be replaced by center c_{in}, where c_{in} and c_{out} represent the indexes of the centers in the array x_{cur}. The *Update* procedure (Algorithm 4) updates the arrays of centers *c1* and *c2* based on the new current solution x_{cur}. Additionally, the presented algorithms use the following functions:

- *nc1(v, in, c1, c2)* - returns the references (closest) center of the user *v*. If the centers *c1(v)* and *c2(v)* are equally distant from the user *v*, the center with closer backup center has the advantage. The new center *in* can also be a backup center;
- *nc2(v, in, c1, c2)* - returns the backup center for the user *v*,
- *nc1Dist(v, p, c1)* - returns the function value for the user *v* and reference center *p*,
- *nc2_1Dist(v, in, c1, c2)* - returns a new function value for the user *v* after its reference center is replaced by the new center *in* in the current solution,
- *nc2_2Dist(v, in, c1, c2)* - returns a new function value for the user *v* after its backup center is replaced by the new center *in* in the current solution,
- *dist(v, u)* - returns the length of the shortest path between the nodes *v* and *u*.

To simplify the pseudocode, in the following algorithms, we omit the *c1* and *c2* arrays from the list of arguments when calling the aforementioned functions.

Algorithm 3: 1-interchange move in the context of the *p*-next center problem

```
Move(x_cur, c_in, c1, c2)
Initialization:
Set z(x_cur(i)) ← 0 for all i = 1, ..., p
Set r(x_cur(i), x_cur(j)) ← 0 for all i = 1, ..., p and j = i+1, ..., p, c_in

Add center:
in ← x_cur(c_in)
For Each user = x_cur(1), …, x_cur(n)
     **calculate function value if center in is a new reference or
       backup center [see Algorithm 3.1]**
     f_in, center ← GetFunctionValueForNewCentarIn(user, in, c1, c2)

     **calculate function value if center in is a new reference or
       backup center and "center" is deleted [see Algorithm 3.2]**
     f_out← GetBackupFunctionValueForNewCentarIn(user, in, center, c1, c2)

     **calculate function value if center in is not a new reference
       center [see Algorithm 3.3]**
     f_cur ← GetFunctionValue(user, in, c1, c2)

     Update r and z values:
     If f_in ≤ f_cur
        **center in is either a new reference or backup center**
        r(in, center) ← r(center, in) ← max(f_in, r(in, center))

        **calculate function value if center in is not a new reference
          center and "center" is deleted [see Algorithm 3.4]**
```

```
        f'out ← GetBackupFunctionValue(user, in, center, c1, c2)

        **in case that "center" is deleted**
        z(center) ← max(min(fout, f'out), z(center))
     Else
        **nc1(user, in) is the reference center; c1(nc1(user, in)) is
          the backup center**
        r(nc1(user, in), c1(nc1(user, in))) ← max(fcur, r(nc1(user, in),
                                                  c1(nc1(user, in))))
        **in case that the reference center is deleted [Algorithm 3.4]**
        f'out ← GetBackupFunctionValue(user, in, nc1(user, in), c1, c2)
        f ← fin if nc1(user, in) ≠ center else fout
        z(nc1(user, in)) ← max(min(f, f'out), z(nc1(user, in)))

        **in case that the backup center is deleted [Algorithm 3.4]**
        f''out ← GetBackupFunctionValue(user, in, c1(nc1(user, in)), c1,
                                                                    c2)
        f ← fin if c1(nc1(user, in)) ≠ center else fout
        z(c1(nc1(user, in))) ← max(min(f, f''out), z(c1(nc1(user, in))))
     End If
End For Each

Best deletion:
cout, fcur ← FindBestDeletion(xcur, r, z, cin) [Algorithm 3.5]

Return fcur, cout.
```

Algorithm 3.1: Calculates function value for the user *u* and the center *in* as a new reference or backup center and also returns counterpart center as a new backup or reference center.

```
GetFunctionValueForNewCentarIn(u, in, c1, c2)
**in as the new reference center**
f ← nc1Dist(u, in) if dist(u, in) ≤ dist(u, nc1(u, in)) else ∞
counterpart_center ← c1(in)

**in as the new backup center**
f' ← dist(u, nc1(u, in)) + dist(nc1(u, in), in)
     if dist(u, nc1(u, in)) ≤ dist(u, in) else ∞
If f > f'
     f ← f'
     counterpart_center ← nc1(u, in)
End If

Return f, counterpart_center.
```

Algorithm 3.2: Calculates function value for the user *u* and *in* as a new reference or backup center if the center *out* is deleted.

```
GetBackupFunctionValueForNewCentarIn(u, in, out, c1, c2)
**"center" is the closest center (not including in), after center out
  has been deleted**
center ← nc1(u, in) if nc1(u, in) ≠ out else nc2(u, in)
**in as the new reference center**
If c1(in) ≠ out
     f' ← nc1Dist(u, in) if dist(u, in) ≤ dist(u, center) else ∞
Else
     f' ← dist(u, in) + dist(in, c2(in))
          if dist(u, in) ≤ dist(u, center) else ∞
End If

**in as the new backup center**
```

```
f'' ← dist(u, center) + dist(center, in)
      if dist(u, center) ≤ dist(u, in) else ∞

Return min(f', f'').
```

Algorithm 3.3: Calculates function value for the user *u* (center *in* might be a new backup center). If the center *in* is the new reference center, it returns *infinity*.

```
GetFunctionValue(u, in, c1, c2)
f ← nc1Dist(u, nc1(u, in)) if dist(u, nc1(u, in)) ≤ dist(u, in) else ∞
Return f.
```

Algorithm 3.4: Calculates function value for the user *u* if the center *out* is deleted (center *in* might be a new backup center). If the center *in* is the new reference center, it returns *infinity*.

```
GetBackupFunctionValue(u, in, out, c1, c2)
If out ≠ nc1(u, in) and out ≠ c1(nc1(u, in))
     **neither the reference nor the backup center is deleted**
     f ← GetFunctionValue(u, in, c1, c2) [Algorithm 3.3]
Else
     If out = nc1(u, in)
        **the reference center is deleted**
        f ← nc2_1Dist(u, in) if dist(u, nc2(u, in)) ≤ dist(u, in) else ∞
     Else
        **the backup center is deleted**
        f ←nc2_2Dist(u, in) if dist(u, nc1(u, in)) ≤ dist(u, in) else ∞
     End If
End If
Return f.
```

Algorithm 3.5: Identifies the center from the current solution, which should be deleted to minimize the function value. Also, it returns the new objective function value.

```
FindBestDeletion(x_cur, r, z, c_in)
f ← ∞
For Each i = {1, ..., p}
     cur ← z(x_cur(i))
     For Each j = {1, ..., p} ∪ {c_in} \ {i}
        If x_cur(i) ≠ c1(x_cur(j))
             cur ← max(r(x_cur(j), c1(x_cur(j))), cur)
        End If
     End For Each
     If f > cur
        f ← cur
        c_out ← i
     End If
End For Each
Return c_out, f.
```

As the steps of Algorithm 3 are not so obvious, we give a brief explanation. Note first that the function value in the current solution extended by the center $in = x_{cur}(c_{in})$ can be calculated as the maximum of all $r(p_i, p_j)$ values, where $r(p_i, p_j)$ represents the maximum value of the pNCP function for all users with allocated p_i as a reference and p_j as a backup center. After initialization, in the *Add center* step, taking into account all users, we check whether the newly-opened center *in* can be allocated to the user (marked with) "*user*" as a new reference or backup center. The function value for the "*user*" in the new solution is also calculated. If the new center *in* becomes a reference or backup center, f_{in} contains the new pNCP function value related to the "*user*". Otherwise, the f_{in} value is ∞. Also, variable

"*center*" denotes the counterpart center, i.e., the reference center if *in* becomes the new backup center and vice versa. It is certain that the newly-added center *in* will not be closed, and therefore it is only necessary to handle the case when the "*center*" is deleted. We also introduce variable f_{out} to hold the function value for the "*user*" when the "*center*" is closed and *in* remains allocated to the "*user*". The f'_{out} value corresponds to the case when the center *in* does not remain allocated to the "*user*".

Having calculated the previous values, including the function value for the "*user*" in the current solution (f_{cur}), the algorithm continues to update the *r* and *z* values. However, note that the f_{cur} value is ∞ if the center *in* is the reference center in the new solution. Providing the new function value f_{in} is better (lower or equal) than f_{cur}, the "*user*" will be allocated a new pair of centers (*in*, *center*). Therefore, the *r(in, center*) is updated if its current value is less than f_{in} (recall that $r(p_i, p_j)$ contains the maximum value). Also, if the *z(center)* value is less than $min(f_{out}, f'_{out})$, it is updated to this value. The $min(f_{out}, f'_{out})$ value corresponds to the new function value in case of potential closure of the "*center*". Otherwise, if f_{cur} is less than f_{in}, the new function value f_{in} is discarded, and the *r* and *z* values are updated based on the previously allocated (reference and backup) centers to the "*user*" and *in* as the potential new backup center. Note that we need to handle cases when either the reference or backup center is closed, i.e., to update *z(the reference center of the user)*, *z(the backup center of the user)* and *r(the reference center, the backup center)* values.

Finally, based on the updated *r* and *z* values, the center to be closed is determined so that the new function value is minimized.

Algorithm 4: Updating the closest and the second-closest center

```
Update(x_cur, c1, c2)
For Each user = x_cur(1), …, x_cur(n)
     c1_dist ← ∞; c1_center ← null
     c2_dist ← ∞; c2_center ← null
     For Each center = x_cur(1), ..., x_cur(p)
         d ← dist(user, center)
         If d < c1_dist or d = c1_dist and
            dist(center, c1(center)) < dist(c1_center, c1(c1_center))
                    c2_center ← c1_center
                    c2_dist ← c1_dist
                    c1_center ← center
                    c1_dist ← d
         Else If d < c2_dist or d = c2_dist and
               dist(center, c1(center)) < dist(c2_center, c1(c2_center))
                    c2_center ← center
                    c2_dist ← d
         End If
     End For Each
     c1(user) ← c1_center
     c2(user) ← c2_center
End For Each
Return c1, c2.
```

Algorithm 5 presents the *Local Search Vertex Substitution* pseudocode. Together with the *c1* and *c2* arrays, the algorithm takes the current solution x_{cur}, the critical user u_c and the current function value f_{cur} as input parameters and updates them if it comes to a better solution. The *Exists Relaxed Distance* procedure (Algorithm 1) checks whether it is

possible to improve the function value by adding a new center and deleting one of the centers from the current solution.

Algorithm 5: The vertex substitution local search for the *p*-next center problem

```
LocalSearchVertexSubstitution(x_cur, c1, c2, u_c, f_cur)
Main loop:
While True
      f' ← ∞
      c_in ← null; c_out ← null

      For Each in = p +1, ..., n
         **Investigate N1(x_cur) neighborhood with center x_cur(in) as a new
           center and find the best deletion**
         If ExistsRelaxedDistance(x_cur, c1, c2, u_c, x_cur(in))
               f, out ← Move(x_cur, in, c1, c2)
               If f < f'
                     f' ← f
                     c_in ← in
                     c_out ← out
               End If
         End If
      End For Each

      If f_cur ≤ f'
         **There is not found improvement in the neighborhood**
         Break Main loop
      End If

      x_cur(c_in) ↔ x_cur(c_out)
      Update(x_cur, c1, c2)
      f_cur ← f'
      u_c ← select user from {x_cur(1), …, x_cur(n)}
           where nc1Dist(user, nc1(user, null)) = f_cur

End While
Return x_cur, u_c, f_cur.
```

In the VNS algorithm (Algorithm 2) which yields the solution to the *p*-next center problem, after the local search phase, if a better (or equal) solution is found, it will become the current solution and the search process will be restarted ($k = 1$). Otherwise, if no better solution is found, the new solution will be discarded and the search will continue in the ($k + 1$)-th neighborhood of the previous solution. If k_{max} is exceeded, k value will be reset to 1 before the next main loop iteration. The main step is repeated until the time limit t_{max} expires. As a result, the algorithm returns the best solution found during execution. If during the search process, the critical user becomes the current solution center and it is not possible to find at least one new center which potentially improves the current solution, the current solution is identified as the exact solution to the *p*-next center problem.