Applied Soft Computing 140 (2023) 110276

Contents lists available at ScienceDirect

Applied Soft Computing

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

Auxiliary data structures and techniques to speed up solving of the *p*-next center problem: A VNS heuristic

Dalibor Ristić [a,*], Nenad Mladenović [b], Mustapha Ratli [c,d], Raca Todosijević [c,d], Dragan Urošević [e,a]

[a] Union University, School of Computing, Knez Mihailova 6, Belgrade, Serbia
[b] Department of Industrial Engineering and Research Center on Digital Supply Chain and Operations Management, Khalifa University, PO Box 127788, Abu Dhabi, United Arab Emirates
[c] Univ. Polytechnique Hauts-de-France, LAMIH, CNRS, UMR 8201, F-59313 Valenciennes, France
[d] INSA Hauts-de-France, F-59313 Valenciennes, France
[e] Mathematical Institute of the Serbian Academy of Sciences and Arts, Knez Mihailova 36, Belgrade, Serbia

ARTICLE INFO

Article history:
Received 5 December 2021
Received in revised form 28 March 2023
Accepted 31 March 2023
Available online 3 April 2023

Keywords:
Location
p-center
p-next center
Heuristics
Variable neighborhood search

ABSTRACT

In this paper we study the *p*-next center problem and propose an adequate solution approach. The *p*-next center problem aims to minimizing the maximum distance from a user to the nearest center plus the distance between the center and its closest center. In this paper we propose a new Variable Neighborhood Search based algorithm to solve the *p*-next center problem. It uses refined local search and shaking procedures as well as auxiliary data structures. The implementation consists in filtering out the candidate centers to enter a solution by considering only ones that potentially decrease the objective function value. The same approach has been applied to the classical *p*-center problem. Here we show that known properties of an efficient implementation of VNS heuristic developed for the *p*-center problem, hold for the new problem as well. More precisely, all the proposals in this work are inspired by other analogous ones used in the literature for similar problems. Hence, the novelty is the adaptation of the known properties that hold for the *p*-center problem to the *p*-next center problem. The performance of the proposed heuristic is assessed on the benchmark instances from the literature as well as newly generated larger instances with 1000, 1500, 2000 and 2500 vertices and instances defined over graphs up to 1000 vertices with different densities. The obtained results clearly demonstrate the effectiveness and efficiency of the proposed algorithm. Hence, the paper shows that the same observations used to solve *p*-center problem may be used to efficiently solve the *p*-next center problem.

Code metadata

Permanent link to reproducible Capsule: https://doi.org/10.24433/CO.0378282.v2.

1. Introduction

The *p*-center problem is a relatively well-known and well-studied facility location problem. It aims to locate *p* identical facilities, called *centers*, on a network to minimize the maximum distance between demand vertices (users) and their closest (nearest) centers. The closest center, assigned to a user, is referred to as *reference center*. The applications of *p*-center problem are mainly related to emergency service locations (determining optimal locations of ambulances, fire stations and police stations) or natural disasters and human-caused disasters. In all of these applications, the worst case service time is extremely important because a prompt action is always sought to respond to requests and save lives. But, what to do if some center fails down? Such situations may occur, for example, in the case of war, earthquakes, tsunamis and hurricanes, when all affected persons instinctively run toward the closest refugee camp, rescue center, hospital etc., but due to high demand/overcrowding the closest center easily becomes unavailable. Another, more recent example is related to the epidemics such as COVID-19, when temporary hospitals and quarantines are quickly filled out and unable to accept new

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: dristic@raf.rs (D. Ristić), nenad@mi.sanu.ac.rs (N. Mladenović), Mustapha.Ratli@uphf.fr (M. Ratli), racatodosijevic@gmail.com (R. Todosijević), draganu@mi.sanu.ac.rs, durosevic@mi.sanu.ac.rs (D. Urošević).

https://doi.org/10.1016/j.asoc.2023.110276
1568-4946/

patients. To resolve such issues one possibility is that all users, whose reference center fails, are re-directed to another open center, called *backup center*, and which is the closest center to that which has become unavailable. In this case, the distance that a user travels until it gets service is equal to the sum of distance from a user to the nearest center, plus the distance between that center and its closest center. The problem that aims to simultaneously optimize the maximum distance from a user to the nearest center plus the distance between that center and its closest center (backup center) is called the p-next center problem (pNCP). This problem is the topic of this paper.

The p-next center problem is defined on an undirected graph $G = (V, E)$, where $V = \{v_1, v_2 \ldots, v_n\}(|V| = n)$ is the set of vertices that represent both the locations of potential centers and users; and $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the edge set. Each edge is assumed to have the distance $d(v_i, v_j)$ that corresponds to the length of the shortest (minimum cost) path that connects v_i and v_j, and therefore the distances satisfy the triangle inequality. The goal of the pNCP is to designate p centers from the given set V so that the maximum distance from a user to its allocated backup center is minimized. The distance from a user to its allocated backup center is calculated as the sum of the distance from the user to its closest (reference) center and the distance from that center to its (backup) center. Let us for a given set $\Pi \subset V$ of p centers define the following sets:

- the set of the closest centers to given vertex $v_i \in V$: $\Pi(v_i) = \arg\min\{d(v_i, \pi_j) | \pi_j \in \Pi\}$;
- the set of centers closest to the center $\pi_j \in \Pi$: $\Pi(\pi_j) = \arg\min\{d(\pi_j, \pi_k) | \pi_k \in \Pi, \pi_j \neq \pi_k\}$.

Then the objective function value for a given set of vertices V and a given set of centers $\Pi \subset V$ is calculated as:

$$f(\Pi) = \max_{v_i \in V} \left\{ \min_{\pi_j \in \Pi(v_i), \pi_k \in \Pi(\pi_j)} \{d(v_i, \pi_j) + d(\pi_j, \pi_k)\} \right\}. \quad (1)$$

In the case that the objective function is calculated with respect to the subset of vertices $\bar{V} \subset V$, the corresponding objective function value of such *restricted problem*, will be denoted as $f(\Pi, \bar{V})$ and is given as:

$$f(\Pi, \bar{V}) = \max_{v_i \in \bar{V}} \left\{ \min_{\pi_j \in \Pi(v_i), \pi_k \in \Pi(\pi_j)} \{d(v_i, \pi_j) + d(\pi_j, \pi_k)\} \right\}. \quad (2)$$

By convention, $f(\Pi, V) = f(\Pi)$.

Contrary to the pNCP problem, the p-center problem (pCP) only considers the distance from the user to its closest (reference) center and therefore the objective function value of the pCP with respect to the given set of vertices V and centers $\Pi \subset V$ is calculated as:

$$FpCP(\Pi, V) = \max_{v_i \in V} \left\{ \min_{\pi_j \in \Pi(v_i)} d(v_i, \pi_j) \right\}. \quad (3)$$

In [1] the p-next center problem has been formulated as a mixed-integer programming (MIP) problem. For that purpose, the authors introduced three sets of variables:

- binary variables y_j that receive value 1 if and only if a center is located at vertex $v_j \in V$;
- binary variables x_{ij} that receive value 1 if and only if the center located at vertex $v_j \in V$ is the closest to the user/center at $v_i \in V$, $i \neq j$; In the case if v_i is a user, the variable indicates the assignment of a center to a user, while if v_i is the center it indicates the assignment of a backup center to the existing center.
- the continuous variable f that reflects the objective function value.

Using the defined variables, they formulated the p-next problem as the following MIP problem:

$$\min f \quad (4)$$

s.t.

$$\sum_{j=1}^{n} y_j = p, \quad (5)$$

$$\sum_{\substack{j=1 \\ j \neq i}} x_{ij} = 1, \ \forall i \in \{1, 2, \ldots, n\}, \quad (6)$$

$$x_{ij} \leq y_j, \ \forall i, j \in \{1, 2, \ldots, n\}, i \neq j, \quad (7)$$

$$y_j + \sum_{\substack{k=1 \\ d(v_i, v_k) > d(v_i, v_j)}}^{n} x_{ik} \leq 1, \ \forall i, j \in \{1, 2, \ldots, n\}, i \neq j, \quad (8)$$

$$f \geq \sum_{\substack{k=1 \\ k \neq j}}^{n} d(v_j, v_k) x_{jk}, \ \forall j \in \{1, 2, \ldots, n\}, \quad (9)$$

$$f \geq d(v_i, v_j)(x_{ij} - y_i) + \sum_{\substack{k=1 \\ k \neq j}}^{n} d(v_j, v_k) x_{jk}, \ \forall i, j \in \{1, 2, \ldots, n\}, i \neq j, \quad (10)$$

$$f \geq 0, x_{ij}, y_j \in \{0, 1\}, \ \forall i, j \in \{1, 2, \ldots, n\}, i \neq j. \quad (11)$$

In the model, the objective function value f is bounded by constraints (9) and (10). Constraints (9) ensure that f is greater or equal to the distance between a reference center and its backup, while constraints (10) guarantee that f is not less than the distance between a user and its backup center. Constraint (5) imposes that exactly p vertices are designated to be the centers. The assignment of users to centers is guaranteed by constraints (6) and (7). Constraints (7) forbid the assignment of users to non-open centers, while constraints (6) require that each vertex (user or center) is assigned to exactly one reference center. Each user/center is assigned to its closest reference center by constraints (8). Finally, variables' domains are stated by constraints (11).

While the literature on the pCP is vast and proposes various solution approaches [2–13], the pNCP, as a recent problem, has started to gain more attention recently. Besides introducing the pNCP, Albareda-Sambola et al. [1] proposed integer programming formulations of the pNCP: two and three-indexed formulations using path variables and a formulation using covering variables. The authors showed that they are capable to solve instances with up to 50 vertices within a reasonable computational effort. They also provided a formal proof that the pNCP is a NP-hard problem.

The first heuristic algorithms for the pNCP were suggested by López-Sánchez et al. [14]. The authors proposed three heuristics: Greedy Randomized Adaptive Search Procedure (GRASP) heuristic, Variable Neighborhood Search (VNS) heuristic and hybrid GRASP-VNS approach combining developed GRASP and VNS heuristics. The proposed GRASP heuristic consists in generating 100 semi-greedy solutions which are subsequently improved by a local search based on the vertex substitution (or *Interchange*) neighborhood. On the other hand, the proposed VNS heuristic follows the basic VNS framework. Starting, from the best solution among 100 semi-greedy solutions, the proposed Basic VNS heuristic applies alternately shaking and local search procedures. The applied local search procedure is the same as the one used in the GRASP heuristic, while the shaking procedure aims to diversify the search by generating a random solution from the kth

neighborhood of the incumbent solution. The kth neighborhood is defined as the set of solutions that may be derived from a given solution by closing k centers in the given solution and opening another k centers. Finally, the hybrid GRASP-VNS approach is derived from the GRASP heuristic by replacing the local search step by the proposed VNS heuristic. The proposed algorithms were evaluated on a set of instances with up to 200 vertices, and as expected, the hybrid algorithm returned the best results. Still, it took much more execution time than the GRASP and VNS executed alone. Comparing GRASP and VNS heuristics, it turned out that they were similar in terms of time consumption. GRASP was a bit faster, but VNS returned slightly better solutions.

Recently, Londe et al. [15] proposed a new hybrid heuristic for solving the pNCP. Their heuristic is a hybrid Biased Random-Key Genetic Algorithm (BRKGA). In that framework, BRKGA is used for creating and maintaining the pool of solutions (population). This standard BRKGA is enhanced by two intensification strategies. The first strategy explores the interchange neighborhood of a solution to possibly reach better quality solutions, while the second strategy, the *implicit path re-linking*, explores paths connecting a base solution and a guide solution, both extracted from the population. In order to avoid premature convergence of BRKGA, due to employed intensification strategies, and to enable better exploration of the solution space, the authors proposed two diversification mechanisms. The first one is the same shaking procedure as in [14], where a random solution from the kth neighborhood is generated by applying k random swap moves. The second one is the full population reset which consists in generating completely new population. The authors tested and implemented three variants of their algorithm: BRKGA-NLS (the local search of the interchange neighborhood is not employed); BRKGA-BI (the local search of the interchange neighborhood with the best improvement strategy is used) and BRKGA-FI (the local search of the interchange neighborhood with the first improvement strategy is used). They concluded that BRKGA-BI was able to provide quality solutions more frequently than the other variants, and recommended it as the best option if the time consumption is not an issue. However, if the time aspect is crucial, and decisions need to be made quickly, they identified BRKGA-NLS as a viable alternative.

In this paper, we suggest a new basic VNS heuristic for the p-next center problem. It differs from the previous one in using more refined local search and shaking procedures. This refined implementation is inspired by work on the p-center problem realized by Mladenović et al. [16] and consists in recognizing, in an advanced way, centers which opening would not improve the current solution. Therefore, our approach filters out the centers considering only ones which opening potentially decreases the objective function value. In addition, the Basic VNS in [14] uses the first improvement search strategy, while our Basic VNS uses the best improvement search strategy. The idea of filtering solutions has been already proposed in [17], where the authors made the first attempt to improve the implementation of Basic VNS presented in [14]. In this paper, we go further and additionally enhance the Basic VNS proposed in [17] by incorporating new data structures to speed up the local search process and present a new more refined shaking procedure. Note that Basic VNS by Ristić et al. [17] uses the same straightforward implementation of the shaking procedure as Basic VNS in [14], while our Basic VNS incorporates a filtering mechanism into the shaking procedure as well. Moreover, here we show that the so-called *Whitaker data structure*, implemented within the interchange (vertex substitution) local search, previously proposed for solving both the p-median problem [18–20], and the p-center problems [16], may be efficiently adapted for solving p-next center problem as well. The *Whitaker data structure*, originally proposed in [20], involves auxiliary arrays that for each vertex store the closest and the second closest center.

Compared to the previous approaches in [14,15,17] we propose a local search that avoids non-promising solutions in a very efficient way. As it will be shown, an iteration of the newly proposed local search has, by the factor n, smaller time complexity than an iteration of the standard local search. In the same spirit we refined the standard shaking procedure used in [14,15,17]. The refinement aims to redirect the search toward promising regions of the solution space without increasing the time complexity compared to the standard shaking procedure. The refinement is the simplest possible, yet effective, and consists in one simple condition that verifies, before executing a random swap move, if it is promising or not. More precisely, our intention was to follow the recent "Less-is-more" approach philosophy [5, 21–24]. We design a Basic VNS algorithm, using the minimum number of search ingredients, but in the most efficient manner. Following this research direction, we prove that the previous data structure in an efficient VNS implementation for solving the p-center problem [16], can be used in the p-next-center problem as well. The novelty is thus theoretical and practical adaptation of known properties that hold for the p-center, to this new problem. In other words, more theory, that allows the simplest possible algorithm. On the other hand, computational results significantly outperform the previous state-of-the-art results from the literature. In addition, we generated larger instances with 1000, 1500, 2000 and 2500 vertices and instances defined over graphs up to 1000 vertices with different densities to assess the performance of the proposed approach on wide range of instances. The obtained results again clearly demonstrate the effectiveness and efficiency of the proposed algorithm. Overall, the contributions of the paper may be summarized as follows:

- The new Basic VNS heuristic for the p-next problem is proposed. It uses refined local search and shaking procedures.
- Known properties that hold for the p-center problem are theoretically and practically extended to the p-next center problem. They include an adaptation of Whitaker data structure within Interchange local search procedure.
- The newly proposed heuristic advances the state-of-the-art results by offering higher number of best-known solutions. In most cases the average time-to-target of proposed heuristic is shorter. In addition, the worst and average solution values in 20 runs are very often better than the state-of-the-art ones.
- This is the first time the instances with up to 2500 vertices are considered for the p-next center. We consider OR-Lib instances which are well-established instances for the p-median and p-center problems, and have up to 900 vertices. In addition new benchmark set of larger instances is proposed and contains instances with up to 2500 vertices.
- Our heuristic shows that it can cope even with such large instances in an efficient and effective way.

The rest of the paper is organized as follows. In Section 2, we explain the steps of the straightforward implementation of the Basic Variable Neighborhood Search heuristic. In the following Section 2.1, we present the enhancements of this implementation by presenting an efficient implementation of the local search within Interchange or Vertex Substitution neighborhood. Section 2.3 contains the refined implementation of the shaking procedure as well as complete pseudo-code of the refined VNS algorithm. In Section 3, we present computational results, and finally, we conclude the paper by summarizing the contributions and giving some future work directions in Section 4.

2. Variable neighborhood search heuristic for the *p*-next center problem

Variable Neighborhood Search metaheuristic was initially introduced by Mladenović and Hansen [25] as a general framework for building heuristics based on systematic changes of neighborhood structures during the search for a (near-) optimal solution. Since then, many heuristics following this recipe have been successfully applied for solving a wide range of optimization and location problems. For example, VNS heuristics have been developed for the solution of the *p*-center problem [16], probabilistic *p*-center problem [26] and *p*-next center problem [14,17].

Basic VNS includes two main phases: the local search phase within one neighborhood structure, and the shaking phase, where the change of neighborhoods occurs. During the local search phase, the current neighborhood is being explored to reach a local optimum, while the goal of the shaking phase is to jump from the current neighborhood to a new one, i.e., escaping from the local optima valleys. In other words, Basic VNS during the local search phase achieves intensification, while jumping to faraway neighborhoods achieves diversification of the search process.

In this section we present the straightforward implementation of the Basic VNS heuristic for the *p*-next center problem. This implementation is also presented in [14], but we repeat it here to show how it may be improved. Within the Basic VNS, the solution of the *p*-next center problem is represented as a set $\Pi = \{\pi_1, \pi_2 \ldots, \pi_p\}$, $|\Pi| = p$, $\Pi \subset V$. Consequently, the kth neighborhood of a solution Π may be defined as:

$$N_k(\Pi) = \{\Pi' | |\Pi'| = p, \Pi' \subset V, |\Pi' \cap \Pi| = p - k\},$$
$$k = 1, 2, \ldots, p.$$

In other words, it represents a set of solutions obtained by replacing k centers from the solution Π by k centers not included in Π. Recall that the set of potential center locations and the set of users coincide, and both are denoted by V, as stated in Introduction.

Based on the previous definitions, Algorithm 1 presents pseudocode of the Basic VNS algorithm for the *p*-next center problem. The algorithm starts by generating an initial solution which is set to be the current incumbent solution Π. After that the main VNS loop starts. It consists in applying alternately, the shaking procedure, the local search procedure and the neighborhood change step in order to enhance the incumbent solution. The shaking procedure aims to diversify the search by generating a random solution from the neighborhood $N_k(\Pi)$. The level of diversification is controlled by the parameter k_{max} which determines the largest possible k value. Initially, k is set to 1, meaning that the solutions from the neighborhood $N_1(\Pi)$ are used to diversify the search. After that, each time an improvement of the incumbent solution occurs, k is reset to 1. Otherwise, its value is incremented by one, meaning that bigger jumps will be made by the shaking procedure $\texttt{Shake}(\Pi, N_k)$ to possibly resolve the current local optima trap. As a local search procedure, the Basic VNS applies the local search within the neighborhood $N_1(\Pi)$. The local search in [14] uses the first improvement search strategy (as soon as a better neighbor solution is detected it is set to be new incumbent solution and search is resumed), while the BVNS we develop in this paper uses the best improvement search strategy (the best improving neighbor solution (if any) is set to be new incumbent solution and the search is resumed). The Basic VNS finishes its work once the maximum allowed CPU time, specified by parameter T_{max}, is reached.

The local search through the neighborhood N_1 examines $p \cdot (n-p)$ possible solutions in each iteration. However, many among them do not improve the current solution. Therefore, it would be beneficial to eliminate these solutions from the search process. This may enable an efficient move to a better solution and therefore significantly speed up the local search process. In the context of *p*-center problems, such speed-up has been done in [16], where the authors called such approach 1-Interchange or Vertex Substitution (VS) heuristic. In this paper, following the ideas from [16], we propose our Vertex Substitution (VS) heuristic applicable to the *p*-next center problem. As already said in the introduction, our aim is to show that the properties that hold for the *p*-center problem are applicable for the *p*-next center problem as well, and that they enable the development of an efficient heuristic for the *p*-next center problem. In addition, we enhance the straightforward Basic VNS (given in Algorithm 1), by developing a refined shaking procedure which does not generate completely random solution from the neighborhood N_k.

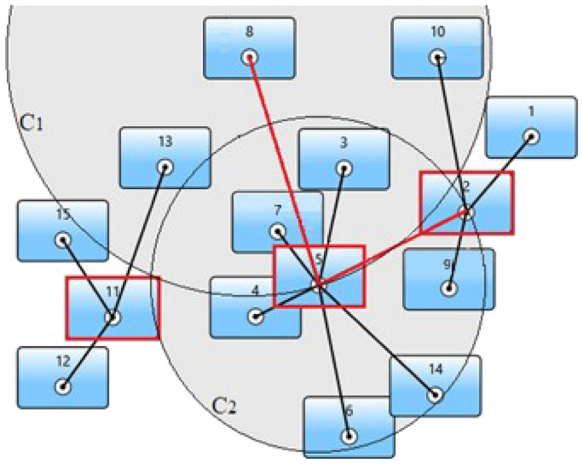
Algorithm 1: Basic VNS for the *p*-next center problem

```
Function BVNS(Π, k_max, T_max)
Π ← Initial_Solution();
repeat
    k ← 1;
    while k ≤ k_max do
        Π' ← Shake (Π, N_k) ;
        Π'' ← Local_Search (Π', N_1) ;
        k ← k + 1;
        if Π'' is better then Π then
            Π ← Π'';
            k ← 1;
    T ← CpuTime();
until T > T_max;
Return Π;
```

2.1. An efficient vertex substitution heuristic for the p-next center problem

Similarly, as observed in the case of the pCP, each solution Π of the *p*-next center problem partitions users into p disjoint subsets, $S_1, S_2, \ldots, S_p$. Each subset set S_i, called a star, contains the users allocated to the same reference center π_i. The distance between the center π_i and its farthest user, plus the distance from center π_i to its closest, i.e., the backup center, $\bar{\pi}_i$ represents the radius of star S_i. The distance is named the radius by analogy to the terminology used in the *p*-center problem. Denoting the radius of each star by $r(S_i)$, it follows that the objective function value of the solution Π is determined by the critical star S_c with the greatest radius, i.e., $f(\Pi) = \max_{i\in\{1,\ldots,p\}}\{r(S_i)\}$. The corresponding center and user, that yield the objective function value, are also called critical and they are denoted by π_c and u_c, respectively.

Consequently, to improve the solution, the largest radius needs to be shortened. This may be accomplished by opening new center which: (i) reduces the distance from the critical user to the reference center and/or; (ii) reduces the distance between the critical center π_c and its backup center. In both cases a new center needs to be opened, and one of the existing centers needs to be closed. The discussed situation is illustrated in Fig. 1. A current solution of the *p*-next center problem with $n = 15$ users and $p = 3$ centers partitions users into three disjoint subsets, S_2, S_5 and S_{11}, represented by the reference centers 2, 5 and 11, respectively. The critical center 5 and the critical user 8 yield the radius $3.94+2.73 = 6.67(cm)$, represented by the red edges $(8, 5)$ and $(5, 2)$. To improve the current solution, the largest distance $d(8, 5) + d(5, 2)$ needs to be shortened (vertex 2 is the backup



(a) Solution

d(vi, vj) cm	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1.63	3.12	5.41	4.28	5.77	4.46	4.80	2.84	2.01	7.50	8.73	6.04	4.54	7.89
2	1.63	0	2.14	3.87	2.73	4.17	3.13	4.37	1.30	2.56	6.08	7.24	5.01	3.07	6.65
3	3.12	2.14	0	2.84	2.00	4.42	1.52	2.38	2.62	2.36	4.54	5.86	2.94	4.05	4.77
4	5.41	3.87	2.84	0	1.14	2.51	1.42	4.26	3.20	5.18	2.35	3.38	2.85	3.25	3.40
5	4.28	2.73	2.00	1.14	0	2.52	1.12	3.94	2.13	4.23	3.43	4.51	3.19	2.64	4.27
6	5.77	4.17	4.42	2.51	2.52	0	3.56	6.45	2.93	6.38	4.36	4.77	5.36	1.58	5.70
7	4.46	3.13	1.52	1.42	1.12	3.56	0	2.92	2.95	3.88	3.06	4.35	2.13	3.74	3.53
8	4.80	4.37	2.38	4.26	3.94	6.45	2.92	0	5.01	3.08	4.85	6.24	2.30	6.35	4.32
9	2.84	1.30	2.62	3.20	2.13	2.93	2.95	5.01	0	3.78	5.54	6.53	5.06	1.77	6.38
10	2.01	2.56	2.36	5.18	4.23	6.38	3.88	3.08	3.78	0	6.83	8.19	4.83	5.54	6.85
11	7.50	6.08	4.54	2.35	3.43	4.36	3.06	4.85	5.54	6.83	0	1.41	2.61	5.47	1.50
12	8.73	7.24	5.86	3.38	4.51	4.77	4.35	6.24	6.53	8.19	1.41	0	3.98	6.13	2.41
13	6.04	5.01	2.94	2.85	3.19	5.36	2.13	2.30	5.06	4.83	2.61	3.98	0	5.82	2.07
14	4.54	3.07	4.05	3.25	2.64	1.58	3.74	6.35	1.77	5.54	5.47	6.13	5.82	0	6.64
15	7.89	6.65	4.77	3.40	4.27	5.70	3.53	4.32	6.38	6.85	1.50	2.41	2.07	6.64	0

(b) Distance matrix

Fig. 1. Example of the p-next center problem with $n = 15$ and $p = 3$; the current solution is $\Pi = \{2, 5, 11\}$ and the critical user $\pi_c = 8$; cluster $S_2 = \{1, 2, 9, 10\}$ has radius $r_2 = 5.29$, reference center 2 and backup center 5, cluster $S_5 = \{3, 4, 5, 6, 7, 8, 14\}$ has radius $r_5 = 6.67$, reference center 5 and backup center 2; cluster $S_{11} = \{11, 12, 13, 15\}$ has radius $r_{11} = 6.04$, reference center 11 and backup center 5. The best improving swap move replaces 11 by 4. The distance matrix is given at the right-side of solution.

center of center 5). Hence, potential centers which opening may reduce the objective function value are ones located inside circle C_1 (of radius $d(8, 5)$ rooted at vertex 8) or centers located inside circle C_2 (of radius $d(5, 2)$ rooted at vertex 5). It should be noted that circle C_1 contains possible locations that may improve reference center positioning, while circle C_2 contains ones that may improve backup center location. For example, if vertex 7 is selected to be a new center, the best choice for the center deletion is center 2 since it yields the lowest objective function value. The new solution is $\Pi = \{5, 7, 11\}$ with the critical user $u_c = 1$, the critical center $\pi_c = 5$, the backup center at vertex 7 and the largest radius $r = d(1, 5) + d(5, 7) = 4.28 + 1.12 = 5.4$. This radius value corresponds to the new objective function value. The user 8, as the critical user from the previous solution, now is included into the cluster $S_7 = \{3, 7, 8, 10, 13\}$ and due to its new reference center 7 and backup center 5, its distance is shortened to $d(8, 7) + d(7, 5) = 2.92 + 1.12 = 4.04$. On the other hand, if the vertex 6 had been selected to be a new center, the objective function value would have increased. Although, the critical user ($p_c = 8$) from the previous solution with the newly assigned pair of centers $(5, 6)$ would shorten its distance ($d(8, 5) + d(5, 6) = 3.94 + 2.52 = 6.46 < 6.67$), the new solution would not be better. The new solution $\Pi = \{5, 6, 11\}$, with the new critical user $\pi_c = 1$, its reference center $\pi_c = 5$ and backup center at vertex 6, would yield the objective function value $d(8, 5) + d(5, 6) = 4.28 + 2.52 = 6.8$, which is worse than the previous 6.67. Therefore, the new solution has been discarded.

Anyway, after all the vertices from the circles C_1 and C_2 have been tried out, it turns out that the vertex 4 is the best choice for the new center. In order to minimize the objective function value, the center 4 should be opened and center 11 closed. In this case, the new solution is $\Pi = \{2, 4, 5\}$; the critical user is $u_c = 10$; its reference and backup centers are 2 and 5, respectively. Such solution yields the maximally improved objective function value $d(10, 2) + d(2, 5) = 2.56 + 2.73 = 5.29$.

The previous observations suggest that to improve a current solution, there is no need to evaluate all solutions in the interchange neighborhood $N_1(\Pi)$. Hence, analogously to [16], we propose an efficient way to discard non-promising solutions and focus only on promising ones. For this purpose, we adopt the following solution representation. Besides a set $\Pi = \{\pi_1, \pi_2, \ldots, \pi_p\}$ of p chosen centers, we use arrays $c1_\Pi$ and $c2_\Pi$ to store, for each vertex, the closest and the second closest center from the set Π. By convention, for a user $v_i \in V$, $c1_\Pi(v_i)$ corresponds to the reference center with respect to solution Π, i.e.,

$$c1_\Pi(v_i) = \arg\min\{d(v_i, \pi_j) | \pi_j \in \Pi\}, \quad \text{for } v_i \in V \setminus \Pi,$$

while for a center $p_i \in \Pi$, $c1_\Pi(p_i)$ corresponds to the backup center from Π, i.e.,

$$c1_\Pi(\pi_i) = \arg\min\{d(\pi_i, \pi_j) | \pi_j \in \Pi, \pi_j \neq \pi_i\}, \quad \text{for } \pi_i \in \Pi.$$

Similarly, we have

$$c2_\Pi(v_i) = \arg\min\{d(v_i, \pi_j) | \pi_j \in \Pi, \pi_j \neq c1_\Pi(v_i)\}, \quad \text{for } v_i \in V \setminus \Pi$$

and

$$c2_\Pi(\pi_i) = \arg\min\{d(\pi_i, \pi_j) | \pi_j \in \Pi, \pi_j \neq \pi_i, \pi_j \neq c1_\Pi(\pi_i)\}, \quad \text{for } \pi_i \in \Pi.$$

Please note that in the case of critical user u_c, $c1_\Pi(u_c)$ equals to p_c and therefore we will use these two notations interchangeably in the rest of the paper.

Determining a promising center to open. Using the introduced notation, we can easily verify whether certain swap move is promising or not. In particular, Algorithm 2 presents the procedure to verify whether opening of center π_{in} is promising or not. The algorithm exploits the fact that a center is promising if its opening shortens the radius associated to the critical user and critical center. Three cases that may lead to an improvement are distinguished:

(1) if centers $c1_\Pi(u_c) = \pi_c$ and $c2_\Pi(u_c)$ are equidistant from the critical user u_c, then the opening of center π_{in}, may enable center $c2_\Pi(u_c)$ to become better option than π_c. This happens if $c2_\Pi(u_c)$ has now closer backup center π_{in},
(2) if center π_{in} is not farther from u_c than π_c, then the radius may be shortened,
(3) if center π_{in} is closer to critical center π_c than $c1_\Pi(\pi_c)$, then the radius may be shortened.

These three cases are elaborated in Algorithm 2, which shows that the verification whether opening of center π_{in} is promising or not,

may be accomplished in the constant time by using arrays $c1_{\Pi}$ and $c2_{\Pi}$.

Algorithm 2: Checking whether there is a better solution if p_{in} is opened as new center

Function ExistsRelaxedDistance($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c, \pi_{in}$)
$f \leftarrow d(u_c, \pi_c) + d(\pi_c, c1_{\Pi}(\pi_c))$;
/*Case 1:*/
if $d(u_c, \pi_c) = d(u_c, c2_{\Pi}(u_c))$ **and** $\min(d(\pi_c, c1_{\Pi}(\pi_c)), d(\pi_c, \pi_{in})) > d(c2_{\Pi}(u_c), \pi_{in})$ **then**
  return True;
/*Case 2:*/
if $d(u_c, \pi_{in}) \leq d(u_c, \pi_c)$ **and** $d(u_c, \pi_{in}) + d(\pi_{in}, c1_{\Pi}(\pi_{in})) < f$ **then**
  return True;
/*Case 3:*/
if $d(u_c, \pi_{in}) \geq d(u_c, \pi_c)$ **and** $d(u_c, \pi_c) + d(\pi_c, \pi_{in}) < f$ **then**
  return True;
Return False;

Determining the best center to close. Once, the promising center π_{in}, to be opened, is detected another center π_{out}, to be closed, needs to be determined. Similarly, as done in [16], we develop the greedy procedure that aims to close a center π_{out} so that the objective function value of solution $\Pi' = \Pi \cup \{\pi_{in}\}$ is deteriorated as least as possible. It should be noted that since the center π_{in} is promising, we have $f(\Pi') < f(\Pi)$. Hence, our aim is to close center π_{out} so that the objective function value $f(\Pi' \setminus \{\pi_{out}\})$ is as close as possible to $f(\Pi')$. In this way we increase chances to obtain a better solution.

For this purposes, two additional data structures are introduced:

- $r'(\pi_i, \pi_j)$ - stores the objective function value for the restricted problem pNCP considering only users that use π_i as the reference and π_j as the backup center, where $\pi_i, \pi_j \in \Pi'$. Note that the values are not computed for all pairs (π_i, π_j), but only for pairs $(\pi_i, c1_{\Pi'}(\pi_i))$, where $c1_{\Pi'}(\pi_i)$ corresponds to the closest center with respect to the set $\Pi' = \Pi \cup \{\pi_{in}\}$. More formally,
$$r'(\pi_i, c1_{\Pi'}(\pi_i)) = \max_{v_j \in V, c1_{\Pi'}(v_j) = \pi_i} d(v_j, \pi_i) + d(\pi_i, c1_{\Pi'}(\pi_i)).$$
It should also be noted that this data structure aims to calculate the radius for each center from the set Π' and therefore it contains the objective function value of the solution $\Pi' = \Pi \cup \{\pi_{in}\}$.
- $z(\pi_i)$ – stores the objective function value of the restricted pNCP problem, where the set of vertices contains only vertices that will change the reference or the backup center if we close center $\pi_i \in \Pi$ (i.e., the set of p centers becomes $\Pi' \setminus \{\pi_i\}$). More formally, let us denote by $\bar{V} = \{v_j \in V | c1_{\Pi'}(v_j) = \pi_i \vee (c1_{\Pi'}(v_j) \neq \pi_i \wedge c1_{\Pi'}(c1_{\Pi'}(v_j)) = \pi_i)\}$, the set of vertices that change their reference or backup center if we exclude vertex $\pi_i \in \Pi$ from the solution Π'. Then, according to Eq. (2), we have:
$$z(\pi_i) = f(\Pi' \setminus \{\pi_i\}, \bar{V}).$$

If these two data structures are available, the objective function value of solution $f(\Pi' \setminus \{\pi_i\})$ may be calculated as:

$$f(\Pi' \setminus \{\pi_i\}) = \max\left\{z(\pi_i), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq \pi_i}} r'(\pi_j, c1_{\Pi'}(\pi_j))\right\}. \quad (12)$$

Consequently, the best center to be closed may be determined as:

$$\pi_{out} = \arg\min_{\pi_i \in \Pi} f(\Pi' \setminus \{\pi_i\}) = \arg\min_{\pi_i \in \Pi}\left\{\max\left\{z(\pi_i), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq \pi_i}} r'(\pi_j, c1_{\Pi'}(\pi_j))\right\}\right\}. \quad (13)$$

Algorithm 3: The fast vertex substitution local search for the p-next center problem

Function FastLocalSearchVertexSubstitution($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c, f_{cur}$)
Improvement $\leftarrow$ **True**;
while Improvement **do**
  $f' \leftarrow \infty$;
  $\pi_{in} \leftarrow null, \pi_{out} \leftarrow null$;
  for *each* $\pi_{in} \in V \setminus \Pi$ **do**
    if ExistsRelaxedDistance($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c, \pi_{in}$) **then**
      $\Pi' \leftarrow \Pi \cup \{\pi_{in}\}$;
      /*Calculate z and r' values */
      $z \leftarrow$ Calculate_z($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c, \pi_{in}$);
      $r' \leftarrow$ Calculate_r'($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c, \pi_{in}$);
      /*determine the best center to close and resulting objective function value */
      $\pi_{out} \leftarrow \arg\min_{\pi_i \in \Pi} f(\Pi' \setminus \{\pi_i\}) = \arg\min_{\pi_i \in \Pi}\left\{\max\left\{z(\pi_i), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq \pi_i}} r'(\pi_j, c1_{\Pi'}(\pi_j))\right\}\right\}$;
      $f'' \leftarrow f(\Pi' \setminus \{\pi_{out}\}) = \max\left\{z(\pi_{out}), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_{out}, c1_{\Pi'}(\pi_j) \neq \pi_{out}}} r'(\pi_j, c1_{\Pi'}(\pi_j))\right\}$;
      if $f'' < f'$ **then**
        $f' \leftarrow f''$;
        $\pi^*_{in} \leftarrow \pi_{in}$;
        $\pi^*_{out} \leftarrow \pi_{out}$;
  if $f_{cur} > f'$ **then**
    $f_{cur} \leftarrow f'$;
    $\Pi \leftarrow \Pi \cup \{\pi^*_{in}\} \setminus \{\pi^*_{out}\}$;
    Update ($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c$);
    Improvement $\leftarrow$ **True**;
  else
    Improvement $\leftarrow$ **False**;
return ($\Pi, c1_{\Pi}, c2_{\Pi}, u_c, \pi_c$);

The outline of the algorithm that examines only promising solutions in the neighborhood $N_1(\Pi)$ and exploits data structures r' and z to determine the best center to be closed is presented in Algorithm 3. The algorithm is named *Fast Local Search Vertex Substitution*. At the input, Algorithm 3 requires solution Π, critical user u_c, critical center π_c as well as associated data structures $c1_{\Pi}$ and $c2_{\Pi}$. The objective function value of the input solution is provided via parameter f_{cur}. Each time a better solution is found all input parameters are updated. Hence, at the output the procedure returns the best encountered solution stored in the input parameters.

$$V' = \left\{ v_i \in V \middle| \left(\begin{array}{c} d(v_i, \pi_{in}) < d(v_i, c1_\Pi(v_i)) \ \vee \\ d(v_i, \pi_{in}) = d(v_i, c1_\Pi(v_i)) \quad \wedge \ d(\pi_{in}, c1_\Pi(\pi_{in})) < d\left(c1_\Pi(v_i), c1_\Pi(c1_\Pi(v_i)) \right) \end{array} \right) \right\}. \tag{14}$$

Box I.

$$V'' = \left\{ v_i \in V \middle| \left(\begin{array}{c} d(c1_\Pi(v_i), \pi_{in}) < d\big(c1_\Pi(v_i), c1_\Pi(c1_\Pi(v_i))\big) \ \wedge \\ d(v_i, c1_\Pi(v_i)) < d(v_i, \pi_{in}) \quad \vee \left(d(v_i, c1_\Pi(v_i)) = d(v_i, \pi_{in}) \wedge d(c1_\Pi(v_i), \pi_{in}) < d(c1_\Pi(\pi_{in}), \pi_{in}) \right) \end{array} \right) \right\}. \tag{16}$$

Box II.

$$V''' = \left\{ v_i \in V \middle| \left(\begin{array}{c} d(v_i, c1_\Pi(v_i)) = d(v_i, c2_\Pi(v_i)) < d(v_i, \pi_{in}) \ \wedge \\ \min\left\{ d(c1_\Pi(v_i), c1_\Pi(c1_\Pi(v_i))), d(c1_\Pi(v_i), \pi_{in}) \right\} > \ d(c2_\Pi(v_i), \pi_{in}) \end{array} \right) \right\}. \tag{18}$$

Box III.

2.2. Theoretical foundations of fast local search vertex substitution algorithm

In this section we discuss the correctness of the precedent algorithm and its complexity. First, we explain how the r' and z values may be computed in an efficient way by using $c1_\Pi$ and $c2_\Pi$ arrays.

Property 1. *Using arrays $c1_\Pi$ and $c2_\Pi$, the r' values in Algorithm 3 may be computed in the time complexity $O(n)$, where n is the number of users.*

Proof. We distinguish four cases:

(1) The opening of new center π_{in} may attract certain users to change their reference centers in one of two following ways: (1) new center π_{in} is closer; or (2) π_{in} and $c1_\Pi(v_i)$ are at the same distance from $v_i \in V$ but the distance between new center and its backup center is smaller. Consequently, the set of such users is defined as Eq. (14) given in Box I. In this case the corresponding r' value is calculated as:

$$r'(\pi_{in}, c1_\Pi(\pi_{in})) = \max_{v_i \in V'} \{ d(v_i, \pi_{in}) + d(\pi_{in}, c1_\Pi(\pi_{in})) \}. \tag{15}$$

(2) The users may keep the allocated reference center but be allocated to a new backup center π_{in}. Hence, we distinguish the set of users given in Box II.
This set enables us to calculate the corresponding r' values, for each pair (π_j, π_{in}), $\pi_j \in \{c1_\Pi(v_i) | v_i \in V''\}$, as:

$$r'(\pi_j, \pi_{in}) = \max_{v_i \in V''} \{ d(v_i, \pi_j) + d(\pi_j, \pi_{in}) | \pi_j = c1_\Pi(v_i) \}. \tag{17}$$

(3) The users may become allocated to $c2_\Pi(v_i)$ and use the center π_{in} as the backup center. This situation occurs if $c1_\Pi(v_i)$ and $c2_\Pi(v_i)$ are equally distant from v_i, center π_{in} is farther from v_i, but center π_{in} enables center $c2_\Pi(v_i)$ to become better reference center (that is, the distance $d(c2_\Pi(v_i), \pi_{in})$ is shorter than any of the distances $d(c1_\Pi(v_i), c1_\Pi(c1_\Pi(v_i)))$, $d(c1_\Pi(v_i), \pi_{in})$). Consequently, we distinguish the set given in Box III. This set enables us to calculate the corresponding r' values, for each pair (π_j, π_{in}), $\pi_j \in \{c2_\Pi(v_i) | v_i \in V'''\}$, as:

$$r'(\pi_j, \pi_{in}) = \max_{v_i \in V'''} \{ d(v_i, \pi_j) + d(\pi_j, \pi_{in}) | \pi_j = c2_\Pi(v_i) \}. \tag{19}$$

(4) The last set includes the users that keep their pair of centers from the solution Π: $V^0 = V \setminus (V' \cup V'' \cup V''')$. In this case r' values are given as:

$$r'(\pi_j, c1_\Pi(\pi_j)) = \max_{v_i \in V^0} \{ d(v_i, \pi_j) + d(\pi_j, c1_\Pi(\pi_j)) | \pi_j = c1_\Pi(v_i) \}. \tag{20}$$

In any of the precedent formulas (15), (17), (19) and (20) the values in the brackets can be calculated in the constant time and therefore the time complexity to calculate all r' values is $O(|V^0| + |V'| + |V''| + |V'''|) = O(|V|) = O(n)$. □

Property 2. *Using arrays $c1_\Pi$ and $c2_\Pi$, the z values in Algorithm 3 may be computed in the time complexity $O(n)$, where n is the number of users.*

Proof. First, we note that for any vertex $v \in V$, the closest center from the set $\Pi_i = \Pi' \setminus \{\pi_i\} = \Pi \setminus \{\pi_i\} \cup \{\pi_{in}\}$, denoted as $c1_{\Pi_i}(v)$, may be determined in the constant time. Namely, the user (center) either keeps the same closest center $c1_\Pi(v)$ it had before, or it is assigned to the closest of $c2_\Pi(v)$ and π_{in}. This observation would be used in the following two cases:

(1) Closing center $\pi_i \in \Pi$ causes a user to lose its reference center. That is $c1_{\Pi'}(v_j) = \pi_i$ which implies $c1_{\Pi'}(v_j) = c1_\Pi(v_j)$. The set of such users is defined as:

$$\overline{V}'_i = \{ v_j \in V | c1_\Pi(v_j) = \pi_i \}.$$

In this case, a user from set $\overline{V}'_i$ is re-assigned to the second closest center $c2_\Pi(v_j)$ or newly added center π_{in}. Let us denote the closest one to v_j as c_{v_j}. Then, the maximum p-next center function value among all users to whom center $\pi_i \in \Pi$ was assigned as a reference center may be

computed as:

$$z'(\pi_i) = \max_{v_j \in \overline{V}_i'} \left\{ d(v_j, c_{v_j}) + d\big(c_{v_j}, c1_{\Pi_i}(c_{v_j})\big) \right\}, \tag{21}$$

where $c1_{\Pi_i}(c_{v_j})$ refers to the center from the set $\Pi_i = \Pi \setminus \{\pi_i\} \cup \{\pi_{in}\}$ which is closest to c_{v_j}. This closest center may be determined in the constant time only examining the centers from the set $C' = \{c1_{\Pi}(c_{v_j}), c2_{\Pi}(c_{v_j}), \pi_{in}\} \setminus \{c_{v_j}, \pi_i\}$. This further implies that in this case $z'(\pi_i)$ is calculated in the time complexity $O(|\overline{V}_i'|)$.

(2) Closing center $\pi_i \in \Pi$ causes a user to lose its backup center. The set of such users is defined as:

$$\overline{V}_i'' = \{v_j \in V | c1_{\Pi'}(c1_{\Pi'}(v_j)) = \pi_i \wedge c1_{\Pi'}(v_j) \neq \pi_i\}.$$

Let us suppose that $c1_{\Pi'}(v_j)$ equals to c_{v_j} and corresponds to the closer of $c1_{\Pi}(v_j)$ and π_{in} to the vertex v_j. If $c_{v_j} = \pi_{in}$ and the current backup center is $c1_{\Pi'}(\pi_{in}) = c1_{\Pi}(\pi_{in}) = \pi_i$, then the new backup center becomes $c2_{\Pi'}(\pi_{in}) = c2_{\Pi}(\pi_{in})$. On the other hand, if $c_{v_j} = c1_{\Pi}(v_j)$ and $c1_{\Pi'}(c_{v_j}) = c1_{\Pi}(c_{v_j}) = \pi_i$, then the new backup center, after closing π_i, will be the closer of $c2_{\Pi}(c_{v_j})$ and π_{in} to c_{v_j}. This implies that the new backup center $c1_{\Pi_i}(c_{v_j})$ may be determined in the constant time from the set $C'' = \{c2_{\Pi}(c_{v_j}), \pi_{in}\} \setminus \{c_{v_j}\}$ as closer one to c_{v_j}.

Therefore, the maximum p-next center function value among all users to whom center $\pi_i \in \Pi$ was assigned as a backup center may be computed in $O(|\overline{V}_i''|)$ time complexity as:

$$z''(\pi_i) = \max_{v_j \in \overline{V}_i''} \left\{ d\big(v_j, c_{v_j}\big) + d\big(c_{v_j}, c1_{\Pi_i}(c_{v_j})\big) \right\}. \tag{22}$$

The observations (1) and (2) imply that the resulting objective function value of all users who lost their reference or backup center is given as:

$$z(\pi_i) = \max\{z'(\pi_i), z''(\pi_i)\}.$$

Hence, we infer that z values may be computed with respect to the users in the time complexity $O(|V|) = O(n)$. □

Having known z and r' values, we show in the next property that the best center to be closed may be found in an efficient way.

Property 3. *The best center to be closed is determined as:*

$$\pi_{out} = \arg\min_{\pi_i \in \Pi} \left\{ \max \left\{ z(\pi_i), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq \pi_i}} r'(\pi_j, c1_{\Pi'}(\pi_j)) \right\} \right\}, \tag{23}$$

where $\Pi' = \Pi \cup \{\pi_{in}\}$. Therefore, it may be determined in time complexity $O(p^2)$, where p is the number of centers.

Proof. Once again we define $\Pi_i = \Pi \setminus \{\pi_i\} \cup \{\pi_{in}\}$ as the solution obtained by closing center π_i and opening center π_{in} and $\Pi' = \Pi \cup \{\pi_{in}\}$ as the solution obtained by opening center π_{in}. In addition, we define by $\overline{V}_i''' = V \setminus \{\overline{V}_i' \cup \overline{V}_i''\}$ the set of users that do not change the reference and backup centers if center $\pi_i \in \Pi$ is closed. The definitions of sets $\overline{V}_i'$ and $\overline{V}_i''$ remain the same as in the previous property. Using this notation, the objective function value of the best solution Π^* is determined as:

$$\begin{aligned}
f(\Pi^*) &= \min_{i \in \{1,2,\ldots p\}} f(\Pi_i) \\
&= \min_{i \in \{1,2,\ldots p\}} \left\{ \max_{v_j \in V} \left\{ d\big(v_j, c1_{\Pi_i}(v_j)\big) + d\big(v_j, c1_{\Pi_i}(c1_{\Pi_i}(v_j))\big) \right\} \right\} \\
&= \min_{i \in \{1,2,\ldots p\}} \left\{ \max_{v_j \in \overline{V}_i' \cup \overline{V}_i'' \cup \overline{V}_i'''} \left\{ d\big(v_j, c1_{\Pi_i}(v_j)\big) + d\big(v_j, c1_{\Pi_i}(c1_{\Pi_i}(v_j))\big) \right\} \right\} \\
&= \min_{i \in \{1,2,\ldots p\}} \left\{ \max \left\{ \begin{array}{l} \max_{v_j \in \overline{V}_i' \cup \overline{V}_i''} \left\{ d\big(v_j, c1_{\Pi_i}(v_j)\big) + d\big(v_j, c1_{\Pi_i}(c1_{\Pi_i}(v_j))\big) \right\}, \\ \max_{v_j \in \overline{V}_i'''} \left\{ d\big(v_j, c1_{\Pi_i}(v_j)\big) + d\big(v_j, c1_{\Pi_i}(c1_{\Pi_i}(v_j))\big) \right\} \end{array} \right\} \right\} \\
&= \min_{i \in \{1,2,\ldots p\}} \left\{ \max \left\{ z(\pi_i), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq \pi_i}} \{r'(\pi_j, c1_{\Pi'}(\pi_j))\} \right\} \right\} \\
&= \min_{p_i \in \Pi} \left\{ \max \left\{ z(\pi_i), \max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq p_i}} \{r'(\pi_j, c1_{\Pi'}(\pi_j))\} \right\} \right\}.
\end{aligned} \tag{24}$$

Note that in Eq. (24) we exploited fact that the value of $\max_{v_j \in \overline{V}_i' \cup \overline{V}_i''} \left\{ d\big(v_j, c1_{\Pi_i}(v_j)\big) + d\big(v_j, c1_{\Pi_i}(c1_{\Pi_i}(v_j))\big) \right\}$ is stored in $z(\pi_i)$. On the other hand the value $\max_{v_j \in \overline{V}_i'''} \left\{ d\big(v_j, c1_{\Pi_i}(v_j)\big) + d\big(v_j, c1_{\Pi_i}(c1_{\Pi_i}(v_j))\big) \right\}$ is actually the largest radius associated to a center who remains in the solution together with its backup center and therefore it may be found as $\max_{\substack{\pi_j \in \Pi', \\ \pi_j \neq \pi_i, c1_{\Pi'}(\pi_j) \neq \pi_i}} \{r'(\pi_j, c1_{\Pi'}(\pi_j))\}$.

Therefore, from the last equation in (24) we conclude that the best center, that should be closed, is determined in the time complexity $O(p^2)$, using auxiliary data structures. □

This property confirms that Algorithm 3, in line 10, properly identifies center π_{out} to be closed. The direct consequence of Eq. (24) is the following corollary, which provides the number of solutions examined by the used greedy approach to identify the best center to be closed.

Corollary. *Using the greedy approach to determine the best center to be closed, the p different solutions from the neighborhood $N_1(\Pi)$ are visited.*

The next property provides overall complexity of `Fast Local Search Vertex Substitution` algorithm (Algorithm 3).

Property 4. *The time complexity of one iteration of* `Fast Local Search Vertex Substitution` *(Algorithm 3) in the worst case is $O(n^2 + p^2 n)$ if the auxiliary data structures are used, n is the number of users and p is the number of centers.*

Proof. In order to determine the time complexity of one iteration of the `Local Search Vertex Substitution` algorithm, it is necessary to determine first the complexity of employed subroutines. Based on Properties 1 and 2, the time complexity of calculating z and r' values is $O(n)$. The best center to be deleted is determined in the complexity $O(p^2)$. The `Update` method updates the arrays $c1_{\Pi}$ and $c2_{\Pi}$ which length is p for each of n users. Hence, its time complexity is $O(pn)$. (Note: If the heap data structure were used to store the second-closest centers, the time complexity would be improved to $O(nlog(p))$, but in that case, the space complexity would increase for $O(pn)$ and the time complexity of `Fast Local Search Vertex Substitution` would not be improved.) The time complexity of the `Exists Relaxed Distance` function is constant. `Fast Local Search Vertex Substitution` uses `Exists Relaxed Distance` and greedy center removal at most $n - p$ times while Update method is used once. Hence, the worst-case time complexity is $O(n^2 + p^2 n) + O(pn) \approx O(n^2 + p^2 n)$. □

In the case of straightforward implementation of the local search within neighborhood $N_1(\Pi)$, we will need to visit $n\times(n-p)$ solutions and to evaluate each solution by formula (1) we need np^2 operations. Hence, the time complexity of straightforward implementation is $O(n^3p^2)$.

On the other hand, if p is significantly less than n, Algorithm 3 has $O(n^2)$ time complexity as stated in the following corollary, while straightforward implementation has time complexity $O(n^3)$.

Corollary. *If $p \ll n$, the time complexity of one iteration of* `Fast Local Search Vertex Substitution` *algorithm (Algorithm 3) is $O(n^2)$, while straightforward implementation has time complexity $O(n^3)$.*

It should be noted that `Fast Local Search Vertex Substitution` algorithm is actually a heuristic approach to explore Interchange neighborhood. Namely, a heuristic approach is applied in any of functions `Exists Relaxed Distance`, `Calculate_z` and `Calculate_r`. This heuristic approach comes from the fact that we do not keep the track of all centers that are equidistant from a user, but we store at most two of them. Preliminary experiments where we kept the track of all equidistant centers revealed that the storing and evaluating all equidistant centers had negative impact on overall performance of our solution approach: the solution process was slowed down and therefore sometimes worse solution values were found within the same time. On the other hand, we recall that our aim is to propose a "less-is-more" solution approach by identifying the least number of ingredients yielding the highest performance. For this reason, we have decided to store at most two equidistant centers (if any) and show in the subsequent sections that even with this reduction our solution approach is very powerful.

2.3. Shaking procedure for the p-next center problem

In addition to the refined local search, we implement a refined shaking procedure as well. The shaking procedure is still based on generating a random solution from the neighborhood $N_k(\Pi)$, but instead of generating completely random solution, our refined procedure generates a semi-greedy solution in each iteration. Namely, to generate a solution from the neighborhood $N_k(\Pi)$, our procedure applies k iterations where in each iteration it exchanges one pair of centers. In each iteration, a center to be opened is chosen so that the current objective function value is improved, while a center to be closed is chosen at random. The outline of shaking procedure is given in Algorithm 4.

Algorithm 4: Refined Shaking Procedure for the p-next center problem

Procedure `RefinedShaking`$(\Pi, c1_\Pi, c2_\Pi, u_c, \pi_c, f_{cur}, k)$
for $j = 1 : k$ **do**
1. Choose center π_{in} to be opened at random;
2. **if** `ExistsRelaxedDistance` $(P, c1_P, c2_P, u_c, \pi_{in})$ **then**
3. $\Pi' \leftarrow \Pi \cup \{\pi_{in}\}$;
4. $r' \leftarrow$ `Calculate_r'`$(\Pi, c1_\Pi, c2_\Pi, u_c, \pi_c, \pi_{in})$;
5. Choose center π_{out} to be closed at random;
6. Calculate $z(\pi_{out})$;
7. $f_{cur} \leftarrow \max\left\{z(\pi_{out}), \max_{\pi_j\in\Pi',\ \pi_j\neq\pi_{out},\ c1_{\Pi'}(\pi_j)\neq\pi_{out}} r'(\pi_j, c1_{\Pi'}(\pi_j))\right\}$;
8. $\Pi \leftarrow \Pi' \setminus \{\pi_{out}\}$;
9. `Update` $(c1_\Pi, c2_\Pi, u_c, \pi_c)$;

Hence, the more refined version of the Algorithm 1 that we implement in this paper is presented in Algorithm 5. It works following the same principles as Algorithm 1, but has more advanced ways of intensifying (see Algorithm 3) and diversifying the search (see Algorithm 4).

Algorithm 5: Refined Basic VNS for the p-next center problem

Function `Refined BVNS`(k_{max}, T_{max})
1. /*random initial solution*/
2. $\Pi \leftarrow$ choose p centers at random;
3. Form arrays $c1_\Pi$ and $c2_\Pi$;
4. Determine critical user u_c and critical center π_c;
5. Calculate objective function value, f_{cur}, of the current solution Π;
6. **repeat**
7. $k \leftarrow 1$;
8. **while** $k \leq k_{max}$ **do**
9. $(\Pi', c1_{\Pi'}, c2_{\Pi'}, u'_c, \pi'_c, f'_{cur}) \leftarrow (\Pi, c1_\Pi, c2_\Pi, u_c, \pi_c, f_{cur})$;
10. `RefinedShaking`$(\Pi', c1_{\Pi'}, c2_{\Pi'}, u'_c, \pi'_c, f'_{cur}, k)$;
11. $(\Pi'', c1_{\Pi''}, c2_{\Pi''}, u''_c, \pi''_c, f''_{cur}) \leftarrow$ `FastLocal SearchVertexSubstitution`$(\Pi', c1_{\Pi'}, c2_{\Pi'}, u'_c, \pi'_c, f'_{cur})$;
12. **if** Π'' *is better then* Π **then**
13. $(\Pi, c1_\Pi, c2_\Pi, u_c, \pi_c, f_{cur}) \leftarrow (\Pi, c1_{\Pi''}, c2_{\Pi''}, u''_c, \pi''_c, f''_{cur})$; $k \leftarrow 1$;
14. $T \leftarrow$ `CpuTime()`;
until $T > T_{max}$;
15. **Return** $(\Pi, c1_\Pi, c2_\Pi, u_c, \pi_c, f_{cur})$;

The algorithm implementation can be downloaded from https://1drv.ms/u/s!AnaX64fUELcfgoF74G7lMuEksN5SGg?e=BFFH8q.

2.4. Less-is-more strategy used in the paper

At the end of this section, we provide more details that highlight what is "less" in our heuristic compared to the previous ones. First of all, we recall that the main idea of the "less-is-more" approach is to use a minimum number of search ingredients, in the most efficient way, so that the developed heuristic is competitive or better than the state-of-the-art approach. In other words, the "less-is-more" methodology highly encourages that we first exploit all techniques that may increase the efficiency of the used set of ingredients before involving new ingredients. For example, the current state-of-the-art approaches hybrid GRASP [14] and hybrid BRKGA [15] use the same local search and shaking procedures, but on top of them they apply other metaheuristic methodologies that make their approaches to be very complex hybrid approaches. As already said, in this paper we follow another philosophy and, as much as possible, try to refine the same local search and shaking procedures used in [14,15]. As a result, we show that the local search within $N_1(\Pi)$ may be performed in an efficient way by avoiding non-promising solutions. Consequently, we come up with the local search procedure whose single iteration has smaller time complexity than the one iteration of the standard local search. In particular the time complexity is reduced by the factor n, where n is the number of vertices. Similarly, we refined the standard shaking procedure. The refinement is the simplest possible, yet effective, and consists in one simple condition that verifies, before executing a random swap move, if it is promising or not. As it will be shown in Section 3.1, this simple refinement enables us to additionally improve the results. It should be noted that this refinement does not increase the time complexity of the shaking procedure. Last but

not least, our heuristic has smaller number of parameters to be adjusted compared to the previous hybrid approaches. Therefore, our heuristic can be considered in this regard more user-friendly.

On the other hand, it is true that our heuristic consumes more memory to store a solution together with auxiliary data structures, compared to the state-of-the-art heuristics. However, this memory consumption equals $O(n)$, where n is the number of vertices, and it is not higher than 200 MB on the largest instances with 2500 vertices. Obviously, such memory consumption is negligible for modern computers. Even if we drastically increase the size of instances, our heuristic will be still executable on any of nowadays computers.

To summarize, in terms of techniques this section proposes two techniques. The first one is to speed up the local search process by screening only promising solutions; while the second technique is the one that enables smarter shaking by hopefully re-directing the search toward promising parts of the solution space instead of performing random jumps. Both techniques are accompanied by auxiliary data structures, which are indeed simple arrays, but which enable us to quickly calculate the objective function values and detect promising zones of the solution space. It should be noted that the inspiration for these techniques and data structures comes from [16,18,20], where authors proposed similar ideas for the p-center problem and p-median problem. Inspired by those ideas, we develop and test, for the first time, auxiliary data structures and techniques for the p-next problem. As it will be shown in Section 3, these techniques and data structures enable us to obtain remarkable results on the benchmark instances.

3. Computational results

In this section, we present the computational results obtained by testing the proposed VNS algorithm for the p-next center problem. The algorithm is implemented using C++ and all tests were carried out on an Intel Core i7-8700K (3.7 GHz) CPU with 32 GB RAM. For testing purposes four different data sets have been used:

- ***OR-Lib instances.*** The data set contains 40 instances from OR-Library [27]. Those instances are well-known benchmark instances used for p-median and p-center problems. In those instances the number of vertices n varies from 100 to 900, while number of sought centers p is between 5 and 200.
- ***Instances from*** [14]. This is the set of small-sized instances that was originally proposed in [14]. It contains 132 test instances derived from OR-Library instances, `pmed1–pmed4` and `pmed6–pmed8`, by taking the first n vertices into consideration. The largest instances contain 200 vertices.
- `rndkreg` ***test instances.*** This set is proposed for the first time in this paper for the p-next center problem. It contains 44 instances with n varying from 1000 to 2500 and p between 5 and 200.
- `rnddnskreg` ***test instances.*** This is the second data set that is proposed in this paper for the p-next center problem. It contains 48 instances with 500–1000 vertices defined over graphs with densities varying from 50% to 80%, and p between 5 and 200.

Please note that Londe et al. [15] developed their own set of benchmark instances. Unfortunately, this data set was not available for us and therefore we did not include it in comparison. However, it should be noted that those instances were derived from OR-Lib instances and consequently the largest instances from the data set of Londe et al. do not have more vertices than the largest instances from OR-Lib data set. On the other hand, here we go even further and propose two data sets that contain instances with larger number of vertices than any instance considered so far for the p-next problem.

The computational experiments are divided into four parts:

- The first part of experiments aims to assess the performance of the proposed refined BVNS algorithm against the Basic VNS proposed in [17] and the Basic VNS that uses the refined local search procedure and the standard shaking procedure of performing a random swap move. The aim is to show benefits of using both refined local search and refined shaking procedures. For this purpose, the OR-Lib test instances are used, the same ones used in [17].
- The second part is devoted to tune parameter k_{max} to identify the best value from the chosen set $\{p/4, p/2, p\}$ of potential values. This experiment is conducted on the set of 30 instances chosen at random from the OR-LIB, `rnddnskreg`, `rndkreg` data sets.
- The third part of experiments compares the proposed refined BVNS and CPLEX MIP solver on the small size instances from [14]. For all those instances, but one, optimal solution values are given in [15]. Hence, the aim of this experiment is to assess the ability of the refined BVNS to attain the optimal solution values.
- The last part of experiments aims to assess the performance of the proposed refined BVNS against state-of-the-art heuristics. The analysis is conducted over all instances from the OR-LIB, `rndkreg` and `rnddnskreg` data sets. As state-of-the-art heuristics we identified the hybrid GRASP heuristic from [14] and the hybrid BRKGA-BI heuristic from [15]. For the sake of brevity they will be referred to as GRASP and BRKGA, in the rest of section. Similarly, the refined BVNS will be referred to as BVNS whenever is suitable. It should be noted that the authors provided us with original source codes of GRASP and BRKGA, and therefore they are executed under same conditions as our BVNS. Consequently, we have had a fair comparison among all heuristics.

On each test instance, all heuristics (refined BVNS, GRASP and BRKGA) has been executed 20 times, each time starting from a different initial solution. Regarding the parameter setting, our BVNS heuristic has two formal parameters, k_{max} – the maximum level of shaking and T_{max} – the maximum CPU time allowed to our heuristic, and their setting will be discussed in subsequent sections. The obtained results of heuristics are assessed in terms of a solution quality (Best, Average, Worst) and time-to-target (the time needed to reach, for the first time, a solution provided at the output). This analysis has been accompanied by adequate statistical analysis. More precisely, to further strengthen analysis, we perform Wilcoxon signed rank test [28] and also derive the performance profiles for both the solution values and times-to-target as suggested in [29].

The Wilcoxon signed rank test is conducted pairwise for any two heuristics H and H'. Let us denote by $Best(H)$, $Avg.(H)$, $Worst(H)$, the set of the best, worst and average solution values, respectively, found by certain heuristic H. Similarly, let us denote by $CPU(H)$, the set of times-to-target consumed by certain heuristic H. Then, for each two H and H' under consideration, we apply Wilcoxon signed rank test on $Best(H)$ and $Best(H')$ values; $Avg.(H)$ and $Avg.(H')$; $Worst(H)$ and $Worst(H')$ and finally on $CPU(H)$ and $CPU(H')$ values.

The performance profiles have been derived relatively to the used set of test instances, and the measure we used to quantify the performance of a heuristic. More precisely, we derive the performance profiles for the Best, Average, and Worst solution values as well as for time-to-target values. For solution values (Best, Average and Worst) the performance profile is derived in

the following way. Let us denote by $\mathcal{H}$, the set of heuristics in comparison, and by M the measure under consideration, which may be the Best solution value, Average Solution or Worst Solution value. Then for each heuristic $H \in \mathcal{H}$ we calculate the ratio R_H^M as: $R_H^M = M_H / \min_{H' \in \mathcal{H}} M_{H'}$. These ratios are then used to derive performance profiles, as described in [29]. The performance profile of heuristic H with respect to metric R_H^M measured over each instance s in a set S is simply the graph of the cumulative distribution function, defined as: $F_M^H(r) = |\{s \in S | R_H^M \leq r\}|/|S|$. In the case of time-to-target, the performance profile of heuristic H with respect to time-to-target $\texttt{CPU\_Time}_H$ measured over each instance s in a set S is the graph of the cumulative distribution function, defined as: $F^H_{\texttt{CPU\_Time}}(r) = |\{s \in S | \texttt{CPU\_Time}_H \leq r\}|/|S|$.

3.1. Assessing the benefits of new local search and shaking procedures

In this section we want to show benefits of using the refined local search and shaking procedures. For this purpose, we compare Basic VNS from [17], refined BVNS presented in Algorithm 5 and BVNS that uses the standard shaking procedure and the refined local search. Note that Basic VNS from [17] uses the standard shaking procedure and the local search procedure that filters solution, without using any auxiliary data structure. Hence, this local search is very close to the standard local search. To differentiate these three variants we will use the following nomenclature: BVNS1 will be Basic VNS from [17]; BVNS2 will be BVNS that uses the standard shaking procedure and the refined local search; and BVNS will be newly proposed refined BVNS.

In [17], BVNS1 was executed on OR-Lib instances with k_{max} set to p and T_{max} set to $5n$, where n is the number of vertices in a considered instance. Hence, for BVNS and BVNS2 we adopt the same value for k_{max}, while we set T_{max} to n. The idea is to show that even with 5 times shorter T_{max}, both BVNS2 and BVNS may outperform BVNS1. The tests are performed on 40 OR-Lib instances as BVNS1 was. On each instance, each variant has been executed 20 times. In Table 1, we report the best, the average and the worst solution values attained in 20 runs (Columns '`Best`', '`Avg.`' and '`Worst`', respectively), the average time-to-target in 20 runs (Columns '`CPU Time`'); and the number of runs (out of 20) in which a heuristic have succeeded to reach the best value reported in the second column (Columns '`# Best`'). The best value corresponds to the best solution value found by one of the three BVNS in comparison. In addition, we provide performance profiles derived with respect to the best, average, worst solution values and time-to-target (see Figs. 2(a)–2(d)). To verify if there is significant difference between each two heuristic with respect to the best, average, worst solution values or time-to-target, we perform Wilcoxon signed rank test and report resulting p values in Table 2.

From the reported results, we observe that the results of BVNS2 are better than the results of BVNS1. BVNS2 has ten times smaller average-time-to-target than BVNS1, but even the average of the worst solution values of BVNS2 is better than the average of the best solution values of BVNS1. The better performance of BVNS2 becomes more evident looking at performance profiles. With respect to any measure (best, average, worst solution value or time-to-target), the curve of BVNS2 is above the curve of BVNS1. In addition, Wilcoxon signed rank test applied to BVNS1 and BVNS2 shows that, at significance level of 5%, there is a significant difference between them. Namely, for each measure (best, average, worst solution value or time-to-target) the resulting p values are less than 0.05. Consequently, we conclude that BVNS2 significantly outperforms BVNS1. Since both heuristic use the same shaking procedure, the obtained results show the clear benefits of using refined local search than the one proposed in [17] (i.e., significantly better solutions in significantly less time). This also confirms the benefits of using the auxiliary data structures.

Comparing BVNS and BVNS2, we observe that BVNS exhibits higher level of stability. The difference between the average of the best solution values and the average of the worst solution values for BVNS is only 0.10, while the same difference for BVNS2 is almost 0.50. In addition, BVNS is only heuristic able to reach all best solution values identified in the second column, while BVNS2 fails to do so in 4 cases. Also, we observe that the average time-to-target of BVNS (that is 19.38) is around 3 times shorter than that of BVNS2 (that is 61.61). Referring to performance profiles, we observe that the curve of BVNS is always above the curve of BVNS2. This signifies that BVNS outperforms BVNS with respect to any measure (best, average, worst solution value or time-to-target). The Wilcoxon signed rank test revealed that, at significance level of 5%, this difference is significant in terms of average, worst solution value or time-to-target. Consequently, we infer that the refined shaking procedure is significantly better option to be coupled with the refined local search than the standard shaking procedure.

3.2. Parameter tuning

In the previous test, k_{max} parameter was set following the recommendations from [17], while T_{max} was set so that is significantly less than the T_{max} from [17]. Now, we set T_{max} to 1800 s, as it is recommended in [15]. Due to this change, we are interested to verify if $k_{max} = p$ is still the best choice or not for our BVNS (Algorithm 5). In this regard, we test three different choices of k_{max}: $\{p/4, p/2, p\}$ and perform the tests on the training set of instances. The training set is a representative subset of 30 instances (around 25% of the total set of instances) that are randomly selected. Hence, the training set contains instances of different size, different p values and different graph densities. It contains 10 instances from each of the sets: OR-Lib, `rndkreg` and `rnddnskreg`. The obtained results are presented in Table 3. For each choice of k_{max} value we report: the best, the average and the worst solution value attained in 20 runs, the average CPU time to attain the final solution for the first time (i.e., the average time-to-target), and the number of times that the algorithm is able to attain the best value reported in the second column. The best value corresponds to the best solution value found under one of three considered parameter settings.

From the reported results we infer that the best average values in terms of best, worst, average solution values are attained by setting k_{max} to p. In addition, this setting enables BVNS from Algorithm 5 to quickly reach the final solution. Namely, the average CPU time of 221.43 s is slightly better compared to other two k_{max} settings. In addition, this setting causes that the best-known solution values (provided in the second column) are attained on average in 16.03 out of 20 runs. Again, this is the highest value compared to the two other settings. The second best setting turns out to be $k_{max} = p/2$, followed by $k_{max} = p/4$.

However, the difference among all settings does not seem significant. The best, worst and average solution values as well as CPU times, under different settings, tend to be very close. To verify this we conduct Wilcoxon signed rank test. The outcomes of Wilcoxon signed rank test are given in Table 4. Since all entries of the table are above 0.05 (the significance level of 5% is considered in Wilcoxon signed rank test), we reject the hypothesis that there is a significant difference for any pair of k_{max} settings. This implies that choosing one of considered k_{max} values does not play critical role in BVNS performance with respect to any of criteria: solution quality or time-to-target. However, since $k_{max} = p$ leads to slightly better results and higher stability of VNS, we decide to fix the value of k_{max} to p in the rest of experiments.

Table 1
Comparison of three BVNS on OR-lib instances.

Test instance	Best	BVNS1					BVNS2					BVNS				
		Solution value			CPU	# Best	Solution value			CPU	# Best	Solution value			CPU	# Best
		Best	Avg.	Worst	time		Best	Avg.	Worst	time		Best	Avg.	Worst	time	
pmed1	166	166	166.30	172	0.50	19	166	166.00	166	0.10	20	166	166	166	0.14	20
pmed2	135	135	136.05	147	100.46	16	135	135.00	135	2.49	20	135	135	135	1.47	20
pmed3	151	151	153.80	164	87.24	15	151	151.00	151	2.27	20	151	151	151	0.49	20
pmed4	118	118	119.35	125	177.84	5	118	118.00	118	4.31	20	118	118	118	0.78	20
pmed5	85	85	85.00	85	38.63	20	85	85.00	85	0.29	20	85	85	85	0.10	20
pmed6	107	107	107.90	111	146.09	14	107	107.00	107	21.92	20	107	107	107	1.91	20
pmed7	84	84	86.00	91	390.52	2	84	84.00	84	16.57	20	84	84	84	10.22	20
pmed8	81	84	86.85	92	332.01	0	81	81.05	82	66.10	19	81	81	81	17.29	20
pmed9	71	71	74.60	75	261.44	2	71	71.00	71	3.98	20	71	71	71	0.41	20
pmed10	70	70	70.00	70	9.74	20	70	70.00	70	0.19	20	70	70	70	0.16	20
pmed11	70	70	70.10	72	168.67	19	70	70.00	70	1.47	20	70	70	70	0.76	20
pmed12	72	72	72.75	79	544.26	16	72	72.00	72	1.70	20	72	72	72	0.85	20
pmed13	47	52	60.95	65	658.49	0	47	48.00	49	102.89	1	47	47	47	17.91	20
pmed14	60	60	60.60	61	515.14	8	60	60.00	60	3.40	20	60	60	60	0.74	20
pmed15	44	44	44.95	48	811.39	13	44	44.00	44	22.62	20	44	44	44	1.68	20
pmed16	54	55	55.10	57	106.52	0	54	54.35	55	98.49	13	54	54	54	57.07	20
pmed17	46	47	49.70	53	791.43	0	46	46.80	47	34.63	4	46	46.35	47	110.24	13
pmed18	50	50	52.40	55	1101.69	6	50	50.00	50	12.39	20	50	50	50	1.67	20
pmed19	32	40	43.50	46	1113.12	0	33	34.80	37	293.75	0	32	32	32	10.61	20
pmed20	40	40	43.30	48	1079.11	4	40	40.00	40	26.26	20	40	40	40	3.05	20
pmed21	48	48	48.80	50	377.36	5	48	48.20	50	52.49	17	48	48.05	49	10.90	19
pmed22	49	52	55.05	58	849.43	0	49	49.45	50	144.02	11	49	49	49	72.78	20
pmed23	32	42	44.40	47	1371.95	0	36	36.75	37	183.51	0	32	32	32	50.16	20
pmed24	33	35	38.40	45	1196.55	0	33	33.15	34	141.54	17	33	33	33	6.89	20
pmed25	44	44	44.00	44	242.45	20	44	44.00	44	4.46	20	44	44	44	1.99	20
pmed26	47	47	47.95	49	905.53	7	47	47.00	47	36.30	20	47	47	47	14.34	20
pmed27	38	40	41.20	43	946.39	0	38	38.55	40	236.00	11	38	38.1	39	124.89	18
pmed28	57	57	57.00	57	20.40	20	57	57.00	57	2.55	20	57	57	57	2.71	20
pmed29	36	36	37.25	42	1461.31	9	36	36.00	36	26.18	20	36	36	36	4.14	20
pmed30	40	40	40.00	40	108.86	20	40	40.00	40	4.24	20	40	40	40	2.65	20
pmed31	35	35	36.95	40	1057.44	7	35	35.00	35	26.29	20	35	35	35	18.77	20
pmed32	72	72	72.00	72	15.29	20	72	72.00	72	4.14	20	72	72	72	4.07	20
pmed33	22	33	34.85	37	1611.42	0	28	28.55	30	442.10	0	22	22.75	23	193.32	5
pmed34	41	41	41.00	41	72.14	20	41	41.00	41	4.33	20	41	41	41	4.39	20
pmed35	36	36	36.85	38	927.56	4	36	36.00	36	48.57	20	36	36	36	17.96	20
pmed36	42	42	42.00	42	223.52	20	42	42.00	42	8.95	20	42	42	42	6.95	20
pmed37	33	33	34.90	38	1634.93	6	33	33.00	33	22.23	20	33	33	33	6.49	20
pmed38	40	40	40.20	41	1102.41	16	40	40.00	40	17.87	20	40	40	40	10.16	20
pmed39	74	74	74.00	74	32.70	20	74	74.00	74	9.02	20	74	74	74	9.29	20
pmed40	23	29	32.10	35	1394.52	0	26	26.75	27	333.57	0	23	23	23	34.15	20
Average	60.63	61.93	63.45	66.23	599.66	9.33	60.98	61.16	61.45	61.61	16.33	60.63	60.66	60.73	20.86	19.38

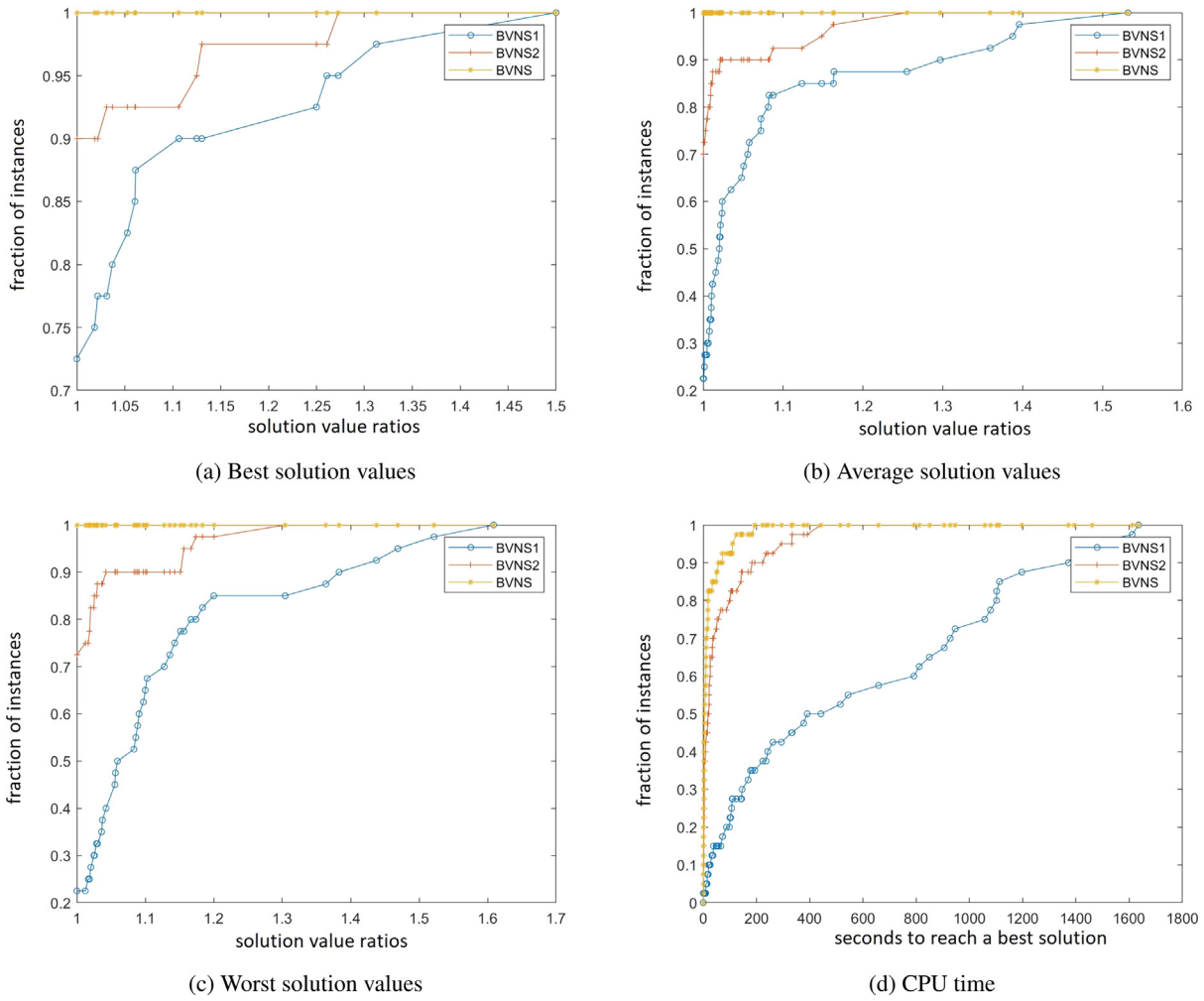
Table 2
Pairwise Wilcoxon test for three BVNS.

Method	Best solution values			Average solution values			Worst solution values			CPU time		
	BVNS1	BVNS2	BVNS	BVNS1	BVNS2	BVNS	BVNS1	BVNS2	BVNS	BVNS1	BVNS2	BVNS
BNS1	–	9.77E−04	9.77E−04	–	1.17E−06	1.17E−06	–	1.69E−06	1.15E−06	–	3.57E−08	3.57E−08
BVNS2	–	–	1.25E−01	–	–	4.88E−04	–	–	9.77E−04	–	–	1.31E−06

3.3. Results on small instances from [14]

The data set from [14] contains 132 small instances. For all instances from this set, but one, optimal solution values are known and published in [15]. Hence, our aim is to verify the capability of the proposed BVNS to retrieve the optimal solutions. The detailed results, per instance, are provided in Appendix 1, while here we provide summary results in Table 5. This table is conceived in the same way as the similar table in [15] and contains the success ratios for BVNS, GRASP and BRKGA. To quantify the success ratio of a heuristic we count the number of optimal solutions it is able to reproduce (Column '# Opt'); calculate the absolute percentage of reproduced optimal solutions as $\frac{\#Opt}{131} \times 100$ (Column '% Opt'); and provide the percentage number of runs a heuristic is able to reach an optimal solution (Column '% Run'). The results for BRKGA and GRASP are directly taken from [15]. To demonstrate fast convergence of our refined BVNS, we present its results for $T_{max} = n$ seconds, where n is number of vertices and $T_{max} = 1800$ seconds. It should be noted that the instances from [14] has from 10 to 200 nodes. Consequently, $T_{max} = n$ is at least 9 times shorter than $T_{max} = 1800$.

According to Table 5, both BVNS heuristics, with different time limits, are able to reach all known optimal solutions, BRKGA does so on 127 instances while GRASP provides optimal solutions on 96 out of 131 instances. In addition, BVNS with $T_{max} = n$, has much higher percentage number of runs in which it provides an optimal solution than GRASP and BRKGA. Extending the time limit to 1800 s, this percentage increases for around 1%, and becomes 10% higher than the percentage number of runs reported for BRKGA. In general there are only 5 instances where BVNS with $T_{max} = n$, did not reach the optimal solution in each of 20 runs, and only 4 such instances for BVNS with $T_{max} = 1800$. Referring to the average time-to-target (see Tables A-1–A-3), we infer that the average time-to-target of BVNS with $T_{max} = n$ is not greater than 81.82 s, which witnesses its efficiency in reproducing an optimal solution.



(a) Best solution values

(b) Average solution values

(c) Worst solution values

(d) CPU time

Fig. 2. Performance profiles for three BVNS on `ORlib` instances.

The only one instance from this set, without known optimal solution, is `pmed6_200_30`. For this instance, CPLEX 12.10 MIP solver returns the solution value of 69 as reported in [15]. On this instance, both BVNS with $T_{max} = n$ and BVNS with $T_{max} = 1800$ are able to provide an improved solution with value of 66 in each run. On the other hand BRKGA and GRASP, executed by us 20 times and time limit of 1800 s, offer the solution which value is 72.

We tried to solve the remaining benchmark instances with CPLEX 22.1 MIP solver imposing the time limit of 8 h (28,800 s) and using MIP formulation presented in this paper. Unfortunately, CPLEX succeeded to provide a feasible solution value on only 16 out of 40 OR-Lib instances (instances `pmed1`–`pmed15` and `pmed20`), while for the remaining 24 OR-Lib instances, and all `rndkreg` and `rnddnskreg` instances failed to provide a feasible solution or at least meaningful lower-bound value. The results obtained on 16 OR-Lib instances, where a feasible solution is attained, are provided in Appendix 2. Among these 16 feasible solutions, only 9 are optimal ones.

3.4. Comparison with state-of-the-art heuristics

This section presents a comparison of the proposed refined BVNS against the state-of-the-art heuristics: the hybrid GRASP from [1] and the hybrid BRKGA from [15]. All three heuristics have been executed by us on the same machine, and run 20 times on each test instance. In each of 20 executions, the maximum allowed time T_{max} for any of heuristic has been set to 1800 s. For testing purpose three data sets are used: OR-Lib, `rndkreg` and `rnddnskreg`. The results are reported in Tables 6, 8 and 10. The first column in each table provides the names of test instances, the next column gives the best-known solution values for each test instances while the remaining ones contain results obtained by three heuristic: the best, the average and the worst solution values in 20 runs (Columns '`Best`', '`Avg.`' and '`Worst`', respectively); the average time-to-target in 20 runs (Columns '`CPU Time`'); and the number of runs (out of 20) in which a heuristic have succeeded to reach best-known value reported in the second column (Columns '`# Best Known`').

3.4.1. Results on OR-Lib instances

From the results on OR-Lib instances, we infer that the proposed refined BVNS exhibits the highest stability. On 37 out of 40 instances, it is able to attain the best-known solution value in each of 20 runs. On the other hand, BRKGA succeeded to do so on 28 out of 40 instances, while GRASP is able to do so on only 10 instances. In addition, BVNS is able to reproduce all best-known solutions, while BRKGA fails to do the same on 7 instances. In this regard GRASP again exhibits the poorest performance failing to reach the best known solution values on 18 instances. Referring to nine optimal solutions found by CPLEX 22.1 MIP solver (Appendix 2, Table A-4), BRKGA and BVNS are able to reach all of them in each run, while GRASP fails to encounter the optimal solution for `pmed20` instance.

Table 3
Tuning k_{max} parameter.

Test instance	Best	$k_{max} = p$					$k_{max} = p/2$					$k_{max} = p/4$				
		Solution value			CPU	# Best	Solution value			CPU	# Best	Solution value			CPU	# Best
		Best	Avg.	Worst	time		Best	Avg.	Worst	time		Best	Avg.	Worst	time	
pmed1	166	166	166.00	166	0.08	20	166	166.00	166	0.12	20	166	166.50	171	0.07	18
pmed10	70	70	70.00	70	0.13	20	70	70.00	70	0.14	20	70	70.00	70	0.13	20
pmed16	54	54	54.00	54	105.30	20	54	54.00	54	195.70	20	55	55.10	57	1.46	0
pmed17	46	46	46.00	46	350.41	20	46	46.25	47	304.71	15	46	46.15	47	513.54	17
pmed20	40	40	40.00	40	3.56	20	40	40.00	40	3.14	20	40	40.00	40	3.31	20
pmed21	48	48	48.05	49	129.68	19	48	48.15	49	21.75	17	48	48.95	51	6.00	8
pmed23	31	31	31.85	32	138.60	3	32	32.00	32	65.26	0	31	31.85	32	172.51	3
pmed27	38	38	38.00	38	244.63	20	38	38.00	38	355.87	20	38	38.05	39	372.01	19
pmed33	22	22	22.25	23	623.52	15	22	22.45	23	584.64	11	22	22.10	23	648.76	18
pmed40	23	23	23.00	23	34.88	20	23	23.00	23	43.02	20	23	23.00	23	36.92	20
rndkreg1	14	14	14.00	14	26.72	20	14	14.00	14	43.58	20	14	14.00	14	36.85	20
rndkreg5	10	10	10.55	11	407.15	9	10	10.40	11	515.85	12	10	10.60	11	361.16	8
rndkreg7	9	9	9.00	9	91.64	20	9	9.00	9	85.89	20	9	9.00	9	95.38	20
rndkreg9	8	8	8.00	8	230.75	20	8	8.00	8	229.17	20	8	8.00	8	255.84	20
rndkreg11	7	7	7.00	7	735.55	20	7	7.05	8	856.14	19	7	7.00	7	716.05	20
rndkreg34	8	8	8.85	9	485.04	3	8	8.80	9	557.29	4	8	8.75	9	583.56	5
rndkreg38	7	7	7.55	8	711.52	9	7	7.50	8	772.67	10	7	7.25	8	816.54	15
rndkreg40	6	6	6.95	7	553.30	1	7	7.00	7	489.89	0	6	6.90	7	524.99	2
rndkreg42	6	6	6.00	6	917.31	20	6	6.00	6	885.64	20	6	6.00	6	874.59	20
rndkreg44	6	6	6.00	6	566.98	20	6	6.00	6	665.55	20	6	6.00	6	578.99	20
rnddnskreg1	7	7	7.00	7	2.47	20	7	7.00	7	2.95	20	7	7.00	7	3.04	20
rnddnskreg5	6	6	6.00	6	4.18	20	6	6.00	6	3.92	20	6	6.00	6	3.85	20
rnddnskreg6	5	5	5.95	6	26.85	1	5	5.90	6	70.88	2	5	5.95	6	21.28	1
rnddnskreg11	4	4	4.00	4	3.45	20	4	4.00	4	3.45	20	4	4.00	4	3.17	20
rnddnskreg12	2	2	2.95	3	69.73	1	2	2.85	3	198.84	3	3	3.00	3	4.01	0
rnddnskreg37	5	5	5.00	5	65.94	20	5	5.00	5	82.25	20	5	5.00	5	71.84	20
rnddnskreg41	5	5	5.00	5	37.04	20	5	5.00	5	34.25	20	5	5.00	5	31.36	20
rnddnskreg42	5	5	5.00	5	40.21	20	5	5.00	5	34.62	20	5	5.00	5	33.99	20
rnddnskreg47	4	4	4.00	4	15.92	20	4	4.00	4	14.31	20	4	4.00	4	17.92	20
rnddnskreg48	3	3	3.00	3	20.34	20	3	3.00	3	21.86	20	3	3.00	3	20.32	20
Average	22.17	22.17	22.365	22.47	221.43	16.03	22.23	22.378	22.53	238.11	15.77	22.23	22.438	22.87	226.98	15.13

Table 4
Pairwise Wilcoxon test for different choices of k_{max} parameter.

k_{max}	Best solution values			Average solution values			Worst solution values			CPU time		
	p	$p/2$	$p/4$	p	$p/2$	$p/4$	p	$p/2$	$p/4$	p	$p/2$	$p/4$
p	–	1.0000	1.0000	–	0.9141	0.7319	–	0.5000	0.1250	–	0.1254	0.9263
$p/2$	–	–	1.0000	–	–	0.6356	–	–	0.1875	–	–	0.5716

Table 5
Comparison on 131 instances from [14] with known optimal solutions.

Method	# Opt	% Opt	% Run
BRKGA	127	96.21	88.11
GRASP	96	72.73	74.42
BVNS ($T_{max} = n$)	131	100.00	97.12
BVNS ($T_{max} = 1800$)	131	100.00	98.14

Looking at performance profiles for the best and average solution values (Figs. 3(a) and 3(b)) we infer that there is no instance on which the best (average) solution value of BVNS is worse than the best (average) solution value of two other heuristics. BRKGA is able to provide the best average solution values on around 70% of instances, while GRASP is able to do so on less than 30% percent of instances. In terms of the quality of the worst solution values, BVNS is again the best. Its curve is on the top of two other curves and on more than 95% instances, its worst solution value is better than the corresponding worst solution values of two other heuristics. The performances of two other heuristics with respect to the worst solution values remain similar as for the average solution values. From all above observations, we may rank BVNS as the best, BRKGA as the second best and GRASP as the worst.

Comparing the average time-to-target we observe that BVNS is the fastest heuristic (average time 49.61), followed by GRASP (the average time 114.77), while BRKGA turns out to be the slowest (the average time 231.21). This observation is further confirmed by the performance profile in Fig. 3(d) where the curve of BVNS is on top of two other curves, and the curve of GRASP is above the curve of BRKGA.

To check if the difference among methods is statistically significant or not, we applied Wilcoxon signed rank test on each pair of heuristics. Table 7 provides the resulting p-values for each pair of heuristics and each measure used (Best Solution Value; Average Solution value; Worst Solution Value; and time-to-target). From the obtained results we infer that at the significance level of 5%, there is a significant difference among methods with respect to Best, Average and Worst Solution Values. In terms of time-to-target, the significant difference has been detected only between BRKGA and BVNS. Consequently, we may conclude that on OR-library instances the proposed BVNS significantly outperforms BRKGA and GRASP in terms of solution quality (Best, Average, and

Table 6
Comparison on ORlib instances.

Test instance	Best known	GRASP					BRKGA					BVNS				
		Solution value			CPU	# Best	Solution value			CPU	# Best	Solution value			CPU	# Best
		Best	Avg.	Worst	time	known	Best	Avg.	Worst	time	known	Best	Avg.	Worst	time	known
pmed1	166	166	166.00	166	0.05	20	166	166.00	166	0.10	20	166	166.00	166	0.08	20
pmed2	135	137	142.25	145	0.14	0	135	135.00	135	2.09	20	135	135.00	135	0.62	20
pmed3	151	151	159.35	170	0.14	2	151	151.00	151	104.30	20	151	151.00	151	0.56	20
pmed4	118	120	128.30	140	0.47	0	118	118.00	118	52.76	20	118	118.00	118	1.16	20
pmed5	85	85	91.40	95	1.14	2	85	85.00	85	1.63	20	85	85.00	85	0.10	20
pmed6	107	107	108.60	110	0.14	7	107	107.00	107	108.73	20	107	107.00	107	3.07	20
pmed7	84	86	90.40	94	0.49	0	84	84.55	85	701.80	9	84	84.00	84	13.76	20
pmed8	81	87	93.50	102	1.78	0	84	84.00	84	81.64	0	81	81.00	81	22.98	20
pmed9	71	75	80.85	85	6.39	0	71	71.00	71	889.38	20	71	71.00	71	0.42	20
pmed10	70	70	70.00	70	13.92	20	70	70.00	70	0.11	20	70	70.00	70	0.13	20
pmed11	70	70	70.75	73	0.28	13	70	70.00	70	3.12	20	70	70.00	70	1.05	20
pmed12	72	72	74.35	78	1.12	9	72	72.00	72	43.39	20	72	72.00	72	0.87	20
pmed13	47	61	65.70	69	7.93	0	51	51.00	51	159.61	0	47	47.00	47	14.39	20
pmed14	60	60	62.35	66	36.47	2	60	60.00	60	4.23	20	60	60.00	60	0.76	20
pmed15	44	48	49.80	53	75.94	0	44	44.00	44	2.29	20	44	44.00	44	1.42	20
pmed16	54	55	55.40	58	0.53	0	54	54.85	55	417.81	3	54	54.00	54	105.30	20
pmed17	46	49	51.60	53	1.83	0	47	47.05	48	777.98	0	46	46.00	46	350.41	20
pmed18	50	53	55.55	59	23.87	0	50	50.00	50	2.83	20	50	50.00	50	1.74	20
pmed19	32	41	43.90	46	75.09	0	33	33.60	34	939.87	0	32	32.00	32	10.50	20
pmed20	40	41	44.10	46	216.35	0	40	40.00	40	3.39	20	40	40.00	40	3.56	20
pmed21	48	48	49.25	51	0.83	5	48	48.00	48	333.50	20	48	48.05	49	129.68	19
pmed22	49	53	56.80	58	3.02	0	49	51.15	52	587.04	2	49	49.00	49	164.52	20
pmed23	31	43	45.35	48	59.11	0	36	37.20	38	927.40	0	31	31.85	32	138.60	3
pmed24	33	36	38.85	42	218.15	0	33	33.00	33	17.99	20	33	33.00	33	6.83	20
pmed25	44	44	44.00	44	427.94	20	44	44.00	44	0.32	20	44	44.00	44	1.89	20
pmed26	47	47	47.70	49	1.15	10	47	47.60	48	925.71	8	47	47.00	47	16.86	20
pmed27	38	40	42.35	45	4.09	0	39	39.75	40	391.88	0	38	38.00	38	244.63	20
pmed28	57	57	57.00	57	111.17	20	57	57.00	57	0.25	20	57	57.00	57	2.58	20
pmed29	36	36	36.75	38	396.01	10	36	36.00	36	1.43	20	36	36.00	36	4.31	20
pmed30	40	40	40.00	40	956.79	20	40	40.00	40	0.41	20	40	40.00	40	2.55	20
pmed31	35	35	36.15	38	1.56	6	35	35.00	35	5.64	20	35	35.00	35	16.93	20
pmed32	72	72	72.00	72	5.54	20	72	72.00	72	0.20	20	72	72.00	72	4.14	20
pmed33	22	33	34.85	37	195.61	0	26	26.95	27	368.81	0	22	22.25	23	623.52	15
pmed34	41	41	41.00	41	836.41	20	41	41.00	41	0.39	20	41	41.00	41	4.35	20
pmed35	36	36	36.45	37	2.04	11	36	36.10	37	872.37	18	36	36.00	36	19.87	20
pmed36	42	42	42.00	42	7.45	20	42	42.00	42	1.00	20	42	42.00	42	7.00	20
pmed37	33	33	34.05	35	374.09	6	33	33.00	33	3.38	20	33	33.00	33	6.09	20
pmed38	40	40	40.00	40	2.54	20	40	40.00	40	1.53	20	40	40.00	40	12.59	20
pmed39	74	74	74.00	74	9.45	20	74	74.00	74	0.24	20	74	74.00	74	9.53	20
pmed40	23	29	30.85	32	513.90	0	23	23.00	23	511.65	20	23	23.00	23	34.88	20
Average:	60.60	62.83	65.09	67.45	114.77	7.08	61.08	61.27	61.40	231.21	16.85	60.60	60.63	60.68	49.61	19.43
Total best-known				22					33					40		

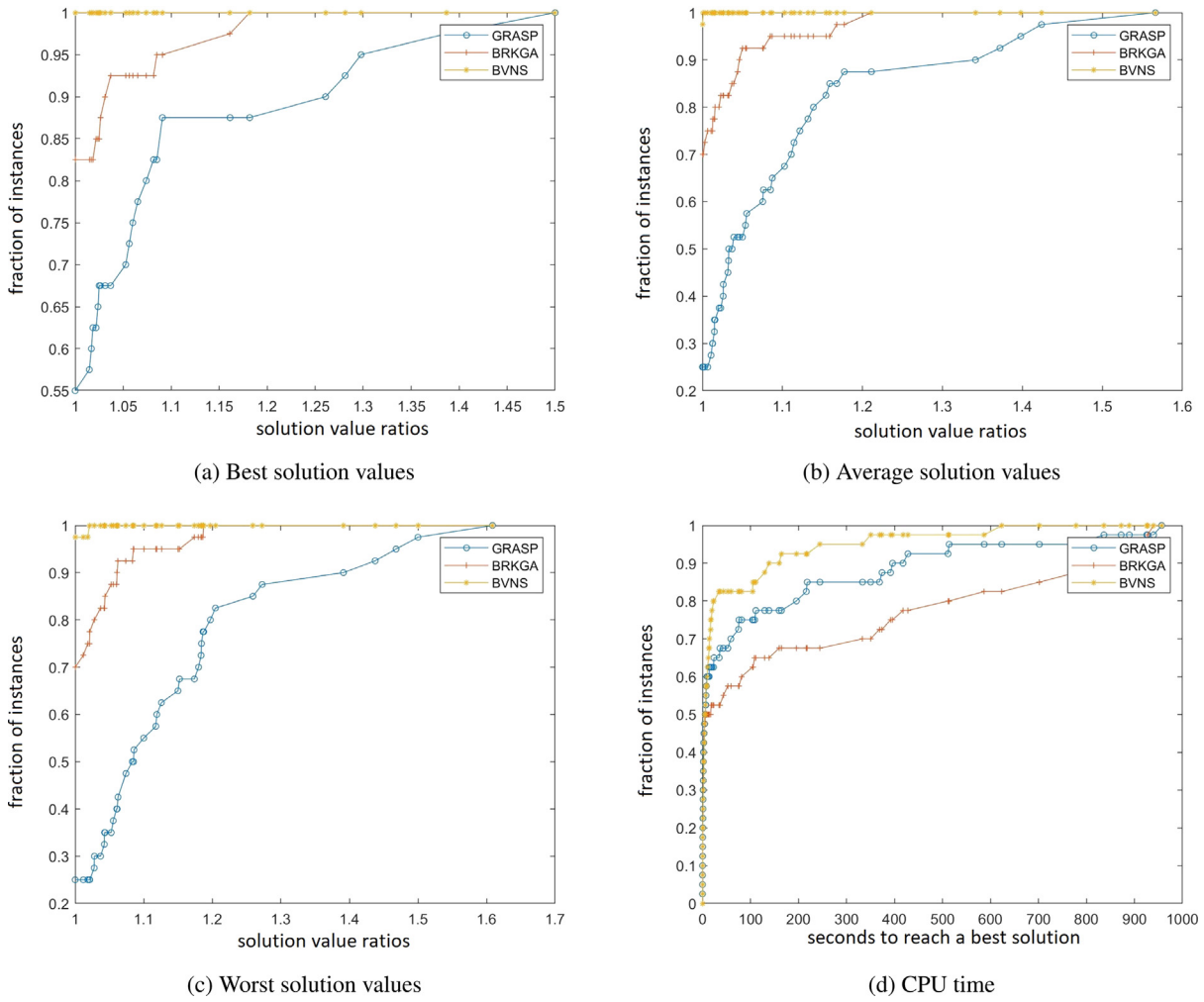
Table 7
Pairwise Wilcoxon test on results on OR-Lib instances.

Method	Best solution values			Average solution values			Worst solution values			CPU time		
	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS
GRASP	–	1.89E−04	1.90E−04	–	1.73E−06	1.73E−06	–	2.46E−06	1.68E−06	–	0.1878	0.5453
BRKGA	–	–	0.0156	–	–	4.88E−04	–	–	1.50E−03	–	–	0.0039

Worst). In addition it significantly outperforms BRKGA in terms of average time-to-target.

3.4.2. Results on `rnddnskreg` instances

On `rnddnskreg` instances the proposed BVNS reaches best-known solution values on 44 out of 48 instances, in each of 20 runs. There is only one instance on which it fails to reproduce the best-known solution in any of 20 runs. BRKGA reaches in total 34 best-known solution values. Among them 29 best-known solution values are reported in each of 20 runs. The performance of GRASP is the poorest. It attains 23 out of 48 best-known solution values and there are only 4 instances on which the best-known solution value is reached in any of 20 runs. Consequently, looking at the performance profiles related to the best solution values (see Fig. 4(a)), BVNS is highly on the top, BRKGA turns out to be the second best and GRASP is the worst. The same tendency may be observed in Figs. 4(b) and 4(c) related to the average and worst solution values. This implies that regarding any criterion (the best, average or worst solution values), the proposed BVNS is the best option. Even more the average of the worst solution values of BVNS is better than the average of the best solution values of BRKGA. Similarly, the average of the worst solution values of BRKGA is better than the average of the best solution values of GRASP. In particular, on 47 out of 48 instances the worst solution values of BVNS are equal or better than the best solution values of BRKGA. On the other hand on all instances, the worst solution values of BRKGA are equal or better than the best solution values of GRASP. The results of Wilcoxon signed rank test (Table 9), with significance level of 5%, imply that indeed there are significant differences among GRASP, BRKGA and BVNS in respect to any



(a) Best solution values

(b) Average solution values

(c) Worst solution values

(d) CPU time

Fig. 3. Performance profiles on OR-Lib instances.

of measures (the best, average or worst solution values). Hence, we may conclude that BVNS significantly outperforms two other heuristics in terms of the solution quality.

Referring to the time-to-target, GRASP turns out to be the slowest with the average time-to-target of 315.67 s. The average times-to-target of BRKGA and BVNS are considerably better: the average time-to-target of BRKGA is 46.52, while that of BVNS is 38.98 s. The results of Wilcoxon signed rank test (Table 9), with significance level of 5%, indicate the there is a significant difference between BVNS and GRASP in terms of time-to-target, while the difference between BVNS and BRKGA, as well as the difference between BRKGA and GRASP, is not statistically significant.

3.4.3. Results on `rndkreg` instances

The results in Table 10 reveal that the `rndkreg` are more challenging than the other instances. This is expected since the number of vertices in `rndkreg` instances is greater than in other instances. Regarding the solution quality we observe that only on 3 instances GRASP has found the best-known solution value, and there is no instance on which it is able to reach the best-known solution value in each of twenty runs. BRKGA behaves better in this regard. It is able to reach 18 best-known solution values, and on 5 instances it is able to reach the best-known solution value in each run. BVNS is the heuristic that establishes the best-known solutions for 41 out of 44 instances and on 25 instances it reaches the best-known solution in each run. On average, it reaches a best known solution in 14.64 runs, BRKGA does the same in 9.41 runs, while GRASP does so in 0.34 (out of 20) runs. In general, even the worst solution values of BVNS are in most cases equal or better than the best solution values of BRKGA. In particular, this happens on 37 out of 44 instances and therefore the average of worst solution values of BVNS is better than the average of best solution values of BRKGA. The same holds when we compare the worst solution values of BRKGA and the best solution values of GRASP. Hence, we may conclude that, regarding solution quality, BVNS is the best option, BRKGA is the second best, while GRASP turns out to be the worst. Our findings are further supported by performance profiles in Figs. 5(a), 5(b) and 5(c). On each of these figures the curve of BVNS is above two other curves, while the curve of BRKGA is between curves of BVNS and GRASP. Moreover, on around 90% of instances BVNS attains the best of the best and average solution values. The best of the best (respectively average) solution values are attained by BRKGA on slightly above 40% (respectively 20%) instances. In this regard, GRASP exhibits very poor performance being able to attain the best of the best solution values on less than 10% of instances and almost none of the best of average solution values. The similar behavior is detected when comparing worst solution values (Fig. 5(c)). On all instances the best of the worst solution values is due to BVNS; on almost 40% the best of worst solution values are attained by

Table 8
Comparison on rnddnskreg instances.

Test instance	Best known	GRASP					BRKGA					BVNS				
		Solution value			CPU	# Best	Solution value			CPU	# Best	Solution value			CPU	# Best
		Best	Avg.	Worst	time	known	Best	Avg.	Worst	time	known	Best	Avg.	Worst	time	known
rnddnskreg1	7	7	7.10	8	0.71	18	7	7.00	7	0.79	20	7	7.00	7	2.47	20
rnddnskreg2	6	7	7.25	8	2.15	0	6	6.00	6	5.46	20	6	6.00	6	15.05	20
rnddnskreg3	5	6	6.45	7	44.96	0	5	5.05	6	331.10	19	5	5.00	5	7.24	20
rnddnskreg4	4	4	4.90	5	491.89	2	4	4.00	4	5.04	20	4	4.00	4	3.29	20
rnddnskreg5	6	6	6.15	7	0.64	17	6	6.00	6	1.04	20	6	6.00	6	4.18	20
rnddnskreg6	5	6	6.55	7	2.05	0	6	6.00	6	0.87	0	5	5.95	6	26.85	1
rnddnskreg7	4	5	5.60	6	42.99	0	5	5.00	5	0.25	0	4	4.00	4	13.66	20
rnddnskreg8	3	4	4.00	4	521.51	0	4	4.00	4	0.51	0	3	3.00	3	8.72	20
rnddnskreg9	5	5	5.90	6	0.62	2	5	5.00	5	5.57	20	5	5.00	5	13.47	20
rnddnskreg10	5	5	5.75	6	2.05	5	5	5.00	5	2.75	20	5	5.00	5	5.07	20
rnddnskreg11	4	5	5.00	5	45.17	0	4	4.95	5	45.00	1	4	4.00	4	3.45	20
rnddnskreg12	2	4	4.00	4	488.65	0	3	3.00	3	21.13	0	2	2.95	3	69.73	1
rnddnskreg13	6	6	6.80	7	0.93	4	6	6.00	6	21.82	20	6	6.00	6	21.93	20
rnddnskreg14	6	6	6.95	7	2.99	1	6	6.00	6	4.32	20	6	6.00	6	11.13	20
rnddnskreg15	4	6	6.00	6	66.06	0	5	5.00	5	164.36	0	4	4.90	5	139.89	2
rnddnskreg16	4	4	4.70	5	774.05	6	4	4.00	4	2.14	20	4	4.00	4	4.71	20
rnddnskreg17	6	6	6.10	7	0.92	18	6	6.00	6	1.68	20	6	6.00	6	6.83	20
rnddnskreg18	5	6	6.05	7	2.98	0	5	5.00	5	50.13	20	5	5.00	5	49.54	20
rnddnskreg19	4	5	5.35	6	62.42	0	4	4.80	5	117.11	4	4	4.00	4	14.13	20
rnddnskreg20	3	4	4.00	4	789.91	0	4	4.00	4	0.62	0	3	3.00	3	14.15	20
rnddnskreg21	5	5	5.25	6	0.85	15	5	5.00	5	1.28	20	5	5.00	5	7.45	20
rnddnskreg22	5	5	5.05	6	2.73	19	5	5.00	5	0.24	20	5	5.00	5	6.68	20
rnddnskreg23	4	5	5.00	5	61.42	0	4	4.25	5	363.70	15	4	4.00	4	5.76	20
rnddnskreg24	3	3	3.95	4	734.86	1	3	3.00	3	16.40	20	3	3.00	3	4.90	20
rnddnskreg25	6	6	6.00	6	1.58	20	6	6.00	6	0.61	20	6	6.00	6	14.92	20
rnddnskreg26	5	6	6.15	7	5.11	0	5	5.00	5	112.87	20	5	5.00	5	518.91	20
rnddnskreg27	4	5	5.90	6	112.20	0	5	5.00	5	216.56	0	4	4.00	4	133.44	20
rnddnskreg28	3	4	4.00	4	1508.31	0	4	4.00	4	0.79	0	3	3.00	3	41.30	20
rnddnskreg29	5	5	5.85	6	1.56	3	5	5.00	5	18.33	20	5	5.00	5	30.35	20
rnddnskreg30	5	5	5.75	6	5.26	5	5	5.00	5	6.59	20	5	5.00	5	23.57	20
rnddnskreg31	4	5	5.00	5	123.74	0	5	5.00	5	0.47	0	4	4.00	4	20.72	20
rnddnskreg32	3	4	4.00	4	1631.05	0	4	4.00	4	0.76	0	3	3.00	3	23.82	20
rnddnskreg33	5	5	5.00	5	1.72	20	5	5.00	5	0.38	20	5	5.00	5	15.86	20
rnddnskreg34	4	5	5.00	5	5.65	0	4	4.55	5	307.36	9	5	5.00	5	16.45	0
rnddnskreg35	4	4	4.65	5	122.52	7	4	4.00	4	28.17	20	4	4.00	4	8.24	20
rnddnskreg36	3	3	3.95	4	1505.11	1	3	3.00	3	75.43	20	3	3.00	3	10.92	20
rnddnskreg37	5	5	5.90	6	2.38	2	5	5.00	5	128.22	20	5	5.00	5	65.94	20
rnddnskreg38	5	6	6.00	6	8.36	0	5	5.00	5	16.94	20	5	5.00	5	70.28	20
rnddnskreg39	4	5	5.10	6	185.45	0	5	5.00	5	2.71	0	4	4.00	4	53.66	20
rnddnskreg40	3	4	4.00	4	1800.01	0	4	4.00	4	1.25	0	3	3.00	3	72.99	20
rnddnskreg41	5	5	5.00	5	2.38	20	5	5.00	5	1.71	20	5	5.00	5	37.04	20
rnddnskreg42	5	5	5.20	6	7.90	16	5	5.00	5	4.11	20	5	5.00	5	40.21	20
rnddnskreg43	4	5	5.00	5	187.28	0	5	5.00	5	0.67	0	4	4.00	4	37.84	20
rnddnskreg44	3	4	4.00	4	1800.01	0	4	4.00	4	1.40	0	3	3.00	3	29.04	20
rnddnskreg45	5	5	5.00	5	2.31	20	5	5.00	5	0.30	20	5	5.00	5	27.71	20
rnddnskreg46	4	5	5.00	5	7.83	0	4	4.00	4	54.12	20	4	4.00	4	81.20	20
rnddnskreg47	4	4	4.20	5	181.11	16	4	4.00	4	0.68	20	4	4.00	4	15.92	20
rnddnskreg48	3	4	4.00	4	1800.01	0	3	3.00	3	89.31	20	3	3.00	3	20.34	20
Average	4.42	4.98	5.28	5.56	315.67	4.96	4.71	4.76	4.81	46.52	13.08	4.44	4.50	4.50	38.98	18.42
Total best-known				23					34					47		

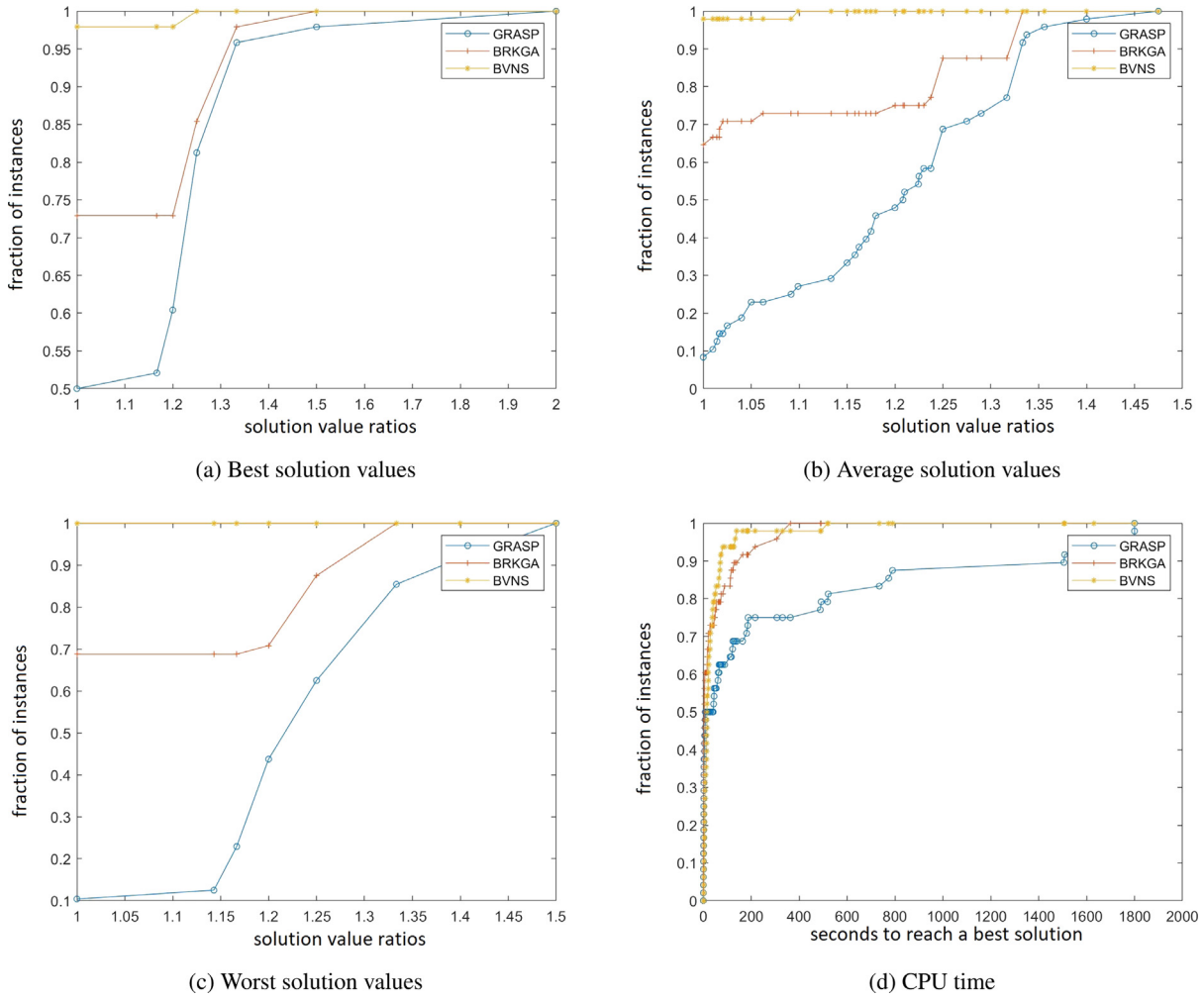
Table 9
Pairwise Wilcoxon test on results on rnddnskreg instances.

Method	Best solution values			Average solution values			Worst solution values			CPU time		
	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS
GRASP	–	2.44E−04	3.66E−06	–	1.65E−07	9.97E−09	–	3.14E−08	8.83E−10	–	0.0777	0.0412
BRKGA	–	–	0.0018	–	–	3.16E−04	–	–	6.10E−05	–	–	0.1662

BRKGA, while GRASP fails to reach the bests of worst solution values on all test instances.

In terms of time-to-target we observe that GRASP heuristic is the slowest one. On most instances the final reported solution is attained at the end of the imposed time limit. This signifies that GRASP has difficulty to cope with large size instances and to quickly explore the large-size solution space. On the other hand, such behavior does not apply to BRKGA and BVNS. Comparing the average times-to-target, BRKGA seems to be faster than BVNS. Looking at performance profiles at Fig. 5(d), this is also confirmed. However, as may be seen at performance profiles, both BRKGA and BVNS do not require more than 15 min to reach a final solution.

To verify if the observed differences among heuristics are statistically significant or not, Wilcoxon signed rank test is conducted. The p-values are given in Table 11. The provided results



(a) Best solution values

(b) Average solution values

(c) Worst solution values

(d) CPU time

Fig. 4. Performance profiles on `rnddnskreg` instances.

show that there is a significant difference, at significance level of 5%, between any pair of heuristics regarding the best, average, worst solution values as well as time-to-target. Taking into account all findings, we conclude that GRASP is not suitable for `rndkreg` instance. It provides significantly poorer solutions and consumes significantly more time-to-target than other approaches. BVNS turns out to be the best option: it provides significantly better solution than any other heuristic. The only drawback of BVNS might be significantly higher time-to–target than that of BRKGA. However, taking into account the remarkable solution values and the highest time-to-target of 15 min to produce a final solution, BVNS can be recommended as the best option to solve `rndkreg` instance of the p-next problem. Namely, the p-next problem arises on strategical/tactical planning level and therefore 15 min to solve large-scale problem instances can be considered as acceptable.

4. Conclusion

This paper studies the p-next center problem. The problem has been introduced as a model of handling humanitarian logistics, and nowadays, during the COVID-19 epidemic, the interest in the problem might significantly increase. The aim is to identify p out of n possible centers, e.g., emergency centers, able to serve all the users, e.g., patients, in a way that the maximum distance from the user's location to the backup center (the center assigned to the user if the reference center is disabled) passing through the reference center (the user's closest center) is minimized among all of the users.

In this paper, we follow the recent "Less-is-more" approach philosophy and design a refined Basic VNS algorithm, using the minimum number of search ingredients in the most efficient manner. As a main contribution, we theoretically and empirically show that the properties and data structures applicable to the p-center problem may be extended to this new problem. In other words, we use more theory which allows the simplest possible algorithm to advance the current state-of-the-art algorithms. Consequently, we show that Interchange neighborhood can be significantly reduced by following sophisticated filtering rules and that the Whitaker data structure, developed originally for solving the p-median problem, works as well for the p-next center problem. In particular, we proposed refined local search and shaking procedures which benefits are empirically evaluated and which are main ingredients of our refined BVNS.

The computational results show that the refined BVNS significantly outperforms the previous state-of-the-art results from the literature in terms of solution quality. It is only heuristic able to reach all optimal solutions for instances from [14]. In addition, on OR-Lib, `rnddnskreg`, and `rndkreg` instances, the refined BVNS is able to reproduce 128 out of 132 best-known solution values, the hybrid BRKGA is able to reproduce 85 out of 132 best-known solution values, while the hybrid GRASP is

Table 10
Comparison on `rndkreg` instances.

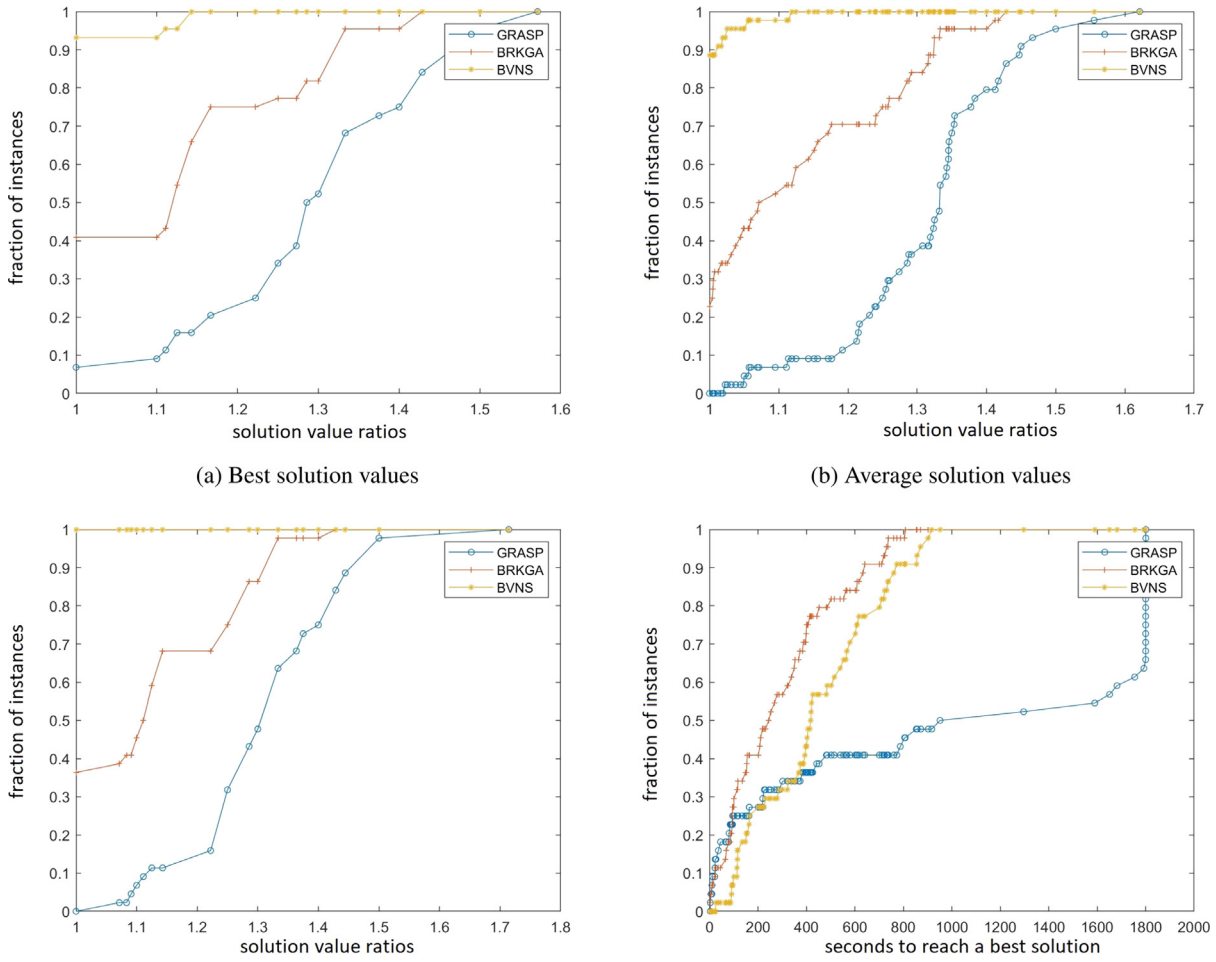
Test instance	Best known	GRASP					BRKGA					BVNS				
		Solution value			CPU time	# Best known	Solution value			CPU time	# Best known	Solution value			CPU time	# Best known
		Best	Avg.	Worst			Best	Avg.	Worst			Best	Avg.	Worst		
rndkreg1	14	14	14.80	15	3.22	4	14	14.10	15	24.15	18	14	14.00	14	26.72	20
rndkreg2	12	14	14.55	15	9.95	0	12	12.05	13	153.69	19	12	12.00	12	116.09	20
rndkreg3	11	14	15.00	16	36.58	0	11	11.70	12	244.92	6	11	11.15	12	374.16	17
rndkreg4	11	14	14.65	15	95.90	0	11	11.00	11	218.69	20	11	11.00	11	152.90	20
rndkreg5	10	13	14.20	15	226.32	0	10	10.95	11	373.96	1	10	10.55	11	407.15	9
rndkreg6	10	14	14.00	14	383.88	0	10	10.95	11	338.37	1	10	10.00	10	115.75	20
rndkreg7	9	12	12.45	13	805.38	0	10	10.00	10	321.65	0	9	9.00	9	91.64	20
rndkreg8	8	11	11.60	12	1296.45	0	9	9.25	10	737.64	0	8	8.00	8	134.61	20
rndkreg9	8	12	12.00	12	1589.97	0	10	10.00	10	268.38	0	8	8.00	8	230.75	20
rndkreg10	7	11	11.00	11	1792.75	0	10	10.00	10	93.44	0	7	7.60	8	394.43	8
rndkreg11	7	11	11.35	12	1800.01	0	10	10.00	10	93.15	0	7	7.00	7	735.55	20
rndkreg12	10	10	11.25	12	6.51	1	10	10.80	11	208.05	4	10	10.10	11	514.66	18
rndkreg13	10	11	12.15	13	22.73	0	10	10.00	10	113.88	20	10	10.00	10	100.73	20
rndkreg14	9	12	12.75	13	81.00	0	9	9.75	10	352.89	5	10	10.00	10	164.46	0
rndkreg15	9	11	12.00	13	219.72	0	9	9.05	10	278.65	19	9	9.00	9	420.17	20
rndkreg16	9	11	11.15	12	482.57	0	9	9.00	9	398.85	20	9	9.00	9	114.26	20
rndkreg17	8	11	11.30	12	853.80	0	9	9.00	9	413.56	0	8	8.40	9	426.07	12
rndkreg18	8	10	10.80	11	1754.86	0	9	9.00	9	563.03	0	8	8.00	8	163.35	20
rndkreg19	7	10	10.00	10	1800.01	0	9	9.00	9	117.63	0	7	7.00	7	417.84	20
rndkreg20	8	10	10.00	10	1800.01	0	9	9.00	9	101.00	0	8	8.00	8	90.71	20
rndkreg21	7	10	10.00	10	1800.64	0	9	9.00	9	79.18	0	7	7.00	7	286.87	20
rndkreg22	6	9	9.80	10	1800.38	0	8	8.90	9	95.01	0	6	6.30	7	902.73	14
rndkreg23	10	10	10.50	11	12.93	10	10	10.05	11	156.26	19	10	10.00	10	201.26	20
rndkreg24	9	10	10.95	12	45.48	0	9	9.00	9	252.70	20	9	9.00	9	367.53	20
rndkreg25	8	10	10.90	11	163.80	0	8	8.05	9	452.14	19	8	8.50	9	539.50	10
rndkreg26	8	10	10.75	11	442.48	0	8	8.15	9	719.13	17	8	8.25	9	616.82	15
rndkreg27	7	10	10.00	10	951.34	0	8	8.20	9	633.57	0	7	7.95	8	403.24	1
rndkreg28	7	10	10.00	10	1681.58	0	8	8.20	9	640.97	0	7	7.85	8	324.27	3
rndkreg29	7	9	9.40	10	1800.01	0	8	8.20	9	609.73	0	7	7.00	7	396.50	20
rndkreg30	6	9	9.00	9	1800.41	0	8	8.00	8	502.15	0	6	6.80	7	421.83	4
rndkreg31	6	9	9.00	9	1800.26	0	8	8.25	9	731.55	0	6	6.65	7	601.77	7
rndkreg32	6	9	9.00	9	1800.61	0	8	8.00	8	3.18	0	6	6.35	7	871.86	13
rndkreg33	6	8	8.80	9	1800.19	0	8	8.00	8	21.94	0	6	6.00	6	773.12	20
rndkreg34	8	9	9.05	10	23.46	0	9	9.00	9	9.84	0	8	8.85	9	485.04	3
rndkreg35	8	9	9.85	10	85.46	0	8	8.00	8	210.84	20	8	8.00	8	607.56	20
rndkreg36	7	9	9.85	10	301.96	0	7	7.15	8	807.63	17	8	8.00	8	722.95	0
rndkreg37	7	9	9.85	10	787.28	0	7	7.85	8	400.65	3	8	8.00	8	580.43	0
rndkreg38	7	9	9.00	9	1651.23	0	8	8.00	8	349.63	0	7	7.55	8	711.52	9
rndkreg39	7	9	9.00	9	1800.01	0	8	8.00	8	387.15	0	7	7.00	7	700.64	20
rndkreg40	6	8	8.95	9	1800.24	0	8	8.00	8	3.15	0	6	6.95	7	553.30	1
rndkreg41	6	8	8.05	9	1800.38	0	7	7.90	8	148.60	0	6	6.00	6	857.07	20
rndkreg42	6	8	8.00	8	1800.13	0	7	7.95	8	69.53	0	6	6.00	6	917.31	20
rndkreg43	6	8	8.00	8	1800.86	0	7	7.95	8	66.23	0	6	6.00	6	760.57	20
rndkreg44	6	7	7.95	8	1800.15	0	7	7.75	8	87.59	0	6	6.00	6	566.98	20
Average	8.00	10.27	10.74	11.07	1484.00	0.34	8.89	9.14	9.41	292.13	5.64	8.07	8.27	8.43	440.06	14.64
Total best-known				3					18					41		

Table 11
Pairwise Wilcoxon test on results on `rndkreg` instances.

Method	Best solution values			Average solution values			Worst solution values			CPU time		
	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS	GRASP	BRKGA	BVNS
GRASP	–	5.93E−08	1.60E−08	–	7.41E−09	7.53E−09	–	5.03E−08	5.26E−09	–	1.23E−04	8.39E−05
BRKGA	–	–	2.41E−05	–	–	1.93E−06	–	–	2.03E−06	–	–	0.0327

able to reproduce only 48 out of 132 best-known solution values. On OR-Lib and `rnddnskreg` instances the refined BVNS is, on average, faster than hybrid BRKGA and GRASP. On the other hand, on `rndkreg` instances the refined BVNS is faster than the hybrid GRASP but slower than the hybrid BRKGA. Therefore, we may conclude that we successfully identified the minimum number of search ingredients that makes our heuristic to be better than more complex hybrid approaches. Therefore, the main conclusion of this work is that the first step in designing a heuristic should be trying to make each search ingredient as efficient as possible before bringing so many different search ingredients that increase complexity.

The possible future research may include the development of VNS algorithms for the other discrete location problems. We are witnesses that the centers could quickly run out of their capacities. Also, in the information age, it is expected to be known in advance if the reference center is unavailable. Therefore, it would be interesting to address the capacitated p-next center and α-neighbor p-center problems, wherein you in advance know whether to go to the reference or any of the backup centers.



(a) Best solution values

(b) Average solution values

(c) Worst solution values

(d) CPU time

Fig. 5. Performance profiles on `rndkreg` instances.

CRediT authorship contribution statement

Dalibor Ristić: Methodology, Software, Formal analysis, Investigation, Data Curating, Writing – original draft. **Nenad Mladenović:** Conceptualization, Validation, Formal analysis, Writing – review & editing, Supervision. **Mustapha Ratli:** Conceptualization, Validation, Writing – review & editing. **Raca Todosijević:** Methodology, Validation, Formal analysis, Resources, Writing – original draft, Writing – review & editing. **Dragan Urošević:** Conceptualization, Methodology, Formal analysis, Investigation, Resources, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request

Acknowledgments

We are more than grateful to the authors of [14,15] who kindly provided us with original source codes of their algorithms. We are also thankful to the reviewers for their valuable comments that helped us to improve our manuscript.

The work of N. Mladenovic has been partially funded by the Science Committee of the Ministry of Education and Science of the Republic of Kazakhstan (Grant No. BR10965172). The work of D. Urošević is partially supported by the Serbian Ministry of Education, Science and Technological Development through Mathematical Institute of the Serbian Academy of Sciences and Arts.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.asoc.2023.110276.

References

[1] M. Albareda-Sambola, Y. Hinojosa, A. Marín, J. Puerto, When centers can fail: a close second opportunity, Comput. Oper. Res. 62 (2015) 145–156.
[2] H. Çalik, M. Labbé, H. Yaman, *p*-Center problems, 2019.
[3] H. Çalik, B.C. Tansel, Double bound method for solving the p-center location problem, Comput. Oper. Res. 40 (2013) 2991–2999.
[4] T. Davidović, D. Ramljak, M. Šelmić, D. Teodorović, Bee colony optimization for the *p*-center problem, Comput. Oper. Res. 38 (2011) 1367–1376.
[5] N. Mladenović, R. Todosijević, Urošević, Less is more: basic variable neighborhood search for minimum differential dispersion problem, Inform. Sci. 326 (2016) 160–171.
[6] S. Elloumi, M. Labbé, Y. Pochet, A new formulation and resolution method for the *p*-center problem, INFORMS J. Comput. 16 (2014) 84–94.

[7] R. Garfinkel, A. Neebe, M. Rao, The *m*-center problem: minimax facility location, Manage. Sci. 23 (10) (1977) 1133–1142.
[8] T. Ilhan, M.C. Pı nar, An efficient exact algorithm for the vertex *p*-center problem, Technical Report, vol. 1, 2001, pp. 209–215.
[9] D.S. Hochbaum, D.B. Shmoys, A best possible heuristic for the *k*-center problem, Math. Oper. Res. 10 (2) (1985) 180–184.
[10] O. Kariv, S.L. Hakimi, An algorithmic approach to network location problems. I: The *p*-centers, SIAM J. Appl. Math. 37 (3) (1979) 513–538.
[11] B. Robič, J. Mihelić, Solving the *k*-center problem efficiently with a dominating set algorithm, J. Comput. Inf. Technol. 13 (3) (2005) 225–234.
[12] W. Pullan, Algorithm for the vertex *p*-center problem, Evol. Comput. 16 (3) (2008) 417–436.
[13] E. Minieka, The *m*-center problem, Soc. Ind. Appl. Math. 12 (1) (1970) 138–139.
[14] A.D. López-Sánchez, J. Sánchez-Oro, A.G. Hernández-Díaz, GRASP and VNS for solving the *p*-next center problem, Comput. Oper. Res. 104 (2019) 295–303.
[15] Mariana A. Londe, Carlos E. Andrade, Luciana S. Pessoa, An evolutionary approach for the p-next center problem, Expert Syst. Appl. 175 (2021) 114728.
[16] N. Mladenović, M. Labbé, P. Hansen, Solving the p-center problem with tabu search and variable neighborhood search, Networks 42 (1) (2003) 48–54.
[17] D. Ristić, N. Mladenović, R. Todosijević, D. Urošević, Filtered variable neighborhood search method for the *p*-next center problem, Int. J. Traffic Transp. Eng. 11 (2) (2021) 294–309.
[18] P. Hansen, N. Mladenović, Variable neighborhood search for the *p*-median, Locat. Sci. 5 (4) (1997) 207–226.
[19] N. Mladenović, J. Brimberg, P. Hansen, J.A. Moreno-Perez, The *p*-median problem: A survey of metaheuristic approaches, European J. Oper. Res. 179 (3) (2007) 927–939.
[20] R.A. Whitaker, A fast algorithm for the greedy interchange for large-scale clustering and median location problems, INFOR: Inf. Syst. Oper. Res. 21 (2) (1983) 95–108.
[21] J. Brimberg, N. Mladenović, R. Todosijević, D. Urošević, Less is more: solving the max-mean diversity problem with variable neighborhood search, Inform. Sci. 382 (2017) 179–200.
[22] L.R. Costa, D. Aloise, N. Mladenović, Less is more: basic variable neighborhood search heuristic for balanced minimum sum-of-squares clustering, Inform. Sci. 415 (2017) 247–253.
[23] K. Goncalves-E-Silva, D. Aloise, S. Xavier-De-Souza, N. Mladenović, Less is more: Simplified nelder-mead method for large unconstrained optimization, YUJOR 28 (2) (2018) 153–169.
[24] M. Mikić, R. Todosijević, D. Urošević, Less is more: General variable neighborhood search for the capacitated modular hub location problem, Comput. Oper. Res. 110 (2019) 101–115.
[25] N. Mladenović, P. Hansen, Variable neighborhood search, Comput. Oper. Res. 24 (11) (1997) 1097–1100.
[26] L.I. Martínez-Merino, M. Albareda-Sambola, A.M. Rodrí guez Chia, The probabilistic p-center problem: planning service for potential customers, European J. Oper. Res. 262 (2) (2017) 509–520–303.
[27] J.E. Beasley, OR-library: distributing test problems by electronic mail, J. Oper. Res. Soc. 41 (11) (1990) 1069–1072.
[28] Frank Wilcoxon, Individual comparisons by ranking methods, Biom. Bull. 6 (1) (1945) 80–83.
[29] Elizabeth D. Dolan, Jorge J. Moré, Benchmarking optimization software with performance profiles, Math. Program. 91 (2) (2002) 201–213.