An iterated greedy algorithm for the obnoxious p-median problem

Osman Gokalp

Department of Computer Engineering, Ege University, Izmir, Turkey

Abstract

The obnoxious p-median problem (OpM) is one of the NP-hard combinatorial optimization problems, in which the goal is to find optimal places to facilities that are undesirable (*e.g.* noisy, dangerous, or pollutant) such that the sum of the minimum distances between each non-facility location and its nearest facility is maximized. In this paper, for the first time in the literature, Iterated Greedy (IG) metaheuristic has been applied at a higher level to solve this problem. A powerful composite local search method has also been developed by combining two fast and effective local search algorithms, namely RLS1 and RLS2, which were previously used to solve the OpM. Comprehensive experiments have been conducted to test the performance of the proposed algorithm using a common benchmark for the problem. The computational results show the effectiveness of the IG algorithm that it can find high-quality solutions in a short time. Based on the set of selected instances, the results also reveal that the developed IG algorithm outperforms most of the state-of-the-art algorithms and contributes to the literature with 5 new best-known solutions.

Keywords: Obnoxious p-median problem, Iterated greedy, Metaheuristics, Combinatorial optimization

Email address: `osman.gokalp@ege.edu.tr` (Osman Gokalp)

1. Introduction

Facility location problems deal with finding optimum places to facilities with respect to given constraints (Farahani and Hekmatfar, 2009). The term facility is used here in a broader context that it may refer to numerous different entities such as schools, bus stops, fire stations, and warehouses (Current et al., 2002). It is generally preferred that the facilities are close to the demand points. However, when facilities are undesirable, or obnoxious, *e.g.* noisy, chemical, nuclear, or pollutant, the goal is to place them as far away from the demand points as possible. In this context, the obnoxious p-median problem (OpM) (Church and Garfinkel, 1978) is defined as to locate p facilities such that the total of minimum distances between each non-facility entity (such as clients or customers) and its nearest facility is maximized. In this way, OpM can be modeled as a p-maxi-sum problem that was proven to be NP-Hard in (Tamir, 1991).

Because OpM is an NP-hard problem, there is no algorithm available that guarantees to find optimum solutions for varying size of p. Therefore, approximation algorithms are preferred to produce acceptable solutions in a reasonable time. Belotti et al. (2007) formulated OpM as a binary linear programming problem and described a Branch and Cut (BC) algorithm (Mitchell, 2002) to solve it. In the same paper, they also improve the performance of BC using eXploring Tabu Search (XTS) (Dell'Amico et al., 1999) approach. Later, Colmenar et al. (2016) first applied a pure heuristic algorithm, based on Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic (Feo and Resende, 1995), to solve the OpM. They showed that GRASP outperformed both BC and XTS algorithms. Then, Herrán et al. (2018) proposed another metaheuristic based on parallel Variable Neighborhood Search (VNS) (Mladenović and Hansen, 1997) along with two simple and fast local search methods. It has been shown that parallel VNS could outperform all the previous algorithms. More recently, Lin and Guan (2018) proposed an algorithm based on a binary Particle Swarm Optimization (PSO) metaheuristic (Eberhart and Kennedy, 1995), and Mladenovic et al. (2019) proposed an algorithm based on a basic VNS for solving the OpM. Despite the important contributions of these studies, the OpM literature is relatively new, and therefore it is considered that producing faster, simpler and more robust algorithms which produce high-quality solutions is still highly valued.

This paper uses Iterated Greedy (IG) algorithm for solving the OpM. As one of the main metaheuristics for solving combinatorial optimization problems, IG consists of two main phases, namely destruction and construction in which solution components are removed and added, respectively. After it was first proposed by Ruiz and Stützle (2007) for solving the permutation flowshop scheduling problem, IG has also been successfully applied to wide range of optimization problems such as traveling salesman problem (Karabulut and Tasgetiren, 2014), job scheduling problem (Arroyo et al., 2019), vehicle routing problem (Nucamendi-Guillén et al., 2018), vertex cover problem (Bouamama et al., 2012), and knapsack problem (García-Martínez et al., 2014). The OpM is another hard combinatorial optimization problem that requires exploring a search space by adding and removing solution components. Therefore, IG algorithm is an ideal candidate for solving the OpM because of its algorithmic structure (i.e. construction/destruction) and robustness.

The main contribution of this paper is to develop an IG algorithm at the master level to solve the OpM for the first time in the literature. Although IG like method was used in (Lin and Guan, 2018) before, it was a very limited version of the algorithm that only consists of single remove and add operations. Also, it was used for just local search step and did not manage the overall optimization process. The second contribution of this work is to develop a composite local search algorithm with a high exploitation capability that combines two simple and fast local search methods.

According to the well-known No Free Lunch Theorem (Wolpert et al., 1997), the performance of an optimization algorithm is highly dependent of a problem type to be solved. In fact, considering all possible problems, the average performance of any pair of algorithms is identical. Therefore, the effectiveness of the proposed IG algorithm has been tested on a common OpM benchmark, which was used previously by all the state-of-the-art algorithms for this problem. The computational results show that, based on the benchmark used, the proposed algorithm outperforms most of the state-of-the-art algorithms in terms of both solution quality and algorithm running time. In addition, it has contributed to the literature by producing 5 new best solutions.

This paper has been organized in the following way. Section 2 gives the mathematical formulation of the OpM and outlines the basic IG and its algorithmic structure. Then, in section 3, the proposed IG algorithm is explained in detail along with its construction rule, composite local search method and solution structure. After that, the experimental framework used in this study, the computational results obtained, and comparison with other algorithms are given in section 4. Finally, section 5 concludes the paper.

2. The background

2.1. Problem formulation

The formal definition of the OpM problem can be given as follows. Let I be the set of clients, J be the set of facilities, and $d_{i,j}$ be the distance between the client $i \in I$ and the facility $j \in J$. Given the objective function $f(\cdot)$, the goal of the problem is to find a set $S \subseteq J$ of size p that maximizes the sum of minimum distances between each client and its nearest facility as follows:

$$\max f(S) = \sum_{i \in I} min\{d_{i,j} : j \in S\} \tag{1}$$

The terms open facility and closed facility are used for the facilities in set S and set $J \setminus S$, respectively.

2.2. Basic iterated greedy algorithm

Iterated Greedy (IG) (Ruiz and Stützle, 2007) is a simple yet powerful metaheuristic algorithm for solving combinatorial optimization problems. IG basically consists of two main phases, namely destruction and construction, which are applied consecutively on a given solution through a number of iterations.

As it is given in Algorithm 1, each iteration starts with the destruction phase in which some part of an incumbent solution is randomly removed, and a partial solution is produced. Then, the missing parts of the partial solution is completed during the construction phase. After that, the local search is optionally applied to the candidate solution for possible improvement. At the final stage of each iteration, the candidate solution is checked whether it will be accepted as a new incumbent solution. These steps are repeated until a termination condition is satisfied (*e.g.* maximum number of iterations, maximum elapsed time).

It should be noted that the general algorithmic structure of IG is similar to that of Iterated Local Search (ILS) (Lourenço et al., 2003) algorithm. In fact, the combination of destruction and construction phases of IG can be seen as a perturbation phase of ILS. However, the difference between the two algorithms is that the perturbation of ILS is only done with random changes in a given neighborhood whereas IG also exploits a constructive heuristic. Therefore, local search is left optional for IG algorithms, which is not necessarily true for ILS (Stützle and Ruiz, 2018).

Algorithm 1: Basic Iterated Greedy algorithm

$S_z \leftarrow$ generate an initial solution ;
$S^* \leftarrow$ apply local search to S_z ; ▷ `optional`
while *termination condition is not satisfied* **do**
    $S_p \leftarrow$ apply destruction to S^* ;
    $S' \leftarrow$ apply construction to S_p ;
    $S' \leftarrow$ apply local search to S' ; ▷ `optional`
    if *acceptance criterion is satisfied* **then**
        $S^* \leftarrow S'$
    end
end
return S^*

3. The proposed iterated greedy algorithm for the OpM

3.1. Pseudocode of the generic algorithm

The outline of the proposed IG algorithm is given in (Alg. 2). First of all, the current solution S is generated randomly by adding one closed facility at a time until the size of the solution reaches p. Additionally, the local search is applied to S and it is stored as the best solution so far, denoted by S^*. Then, the algorithm tries to improve the S^* in its main loop until the maximum number of iterations (MAX_ITER) is reached.

At the beginning of each iteration, the destruction size d is calculated proportionally to the solution size p using the parameter $d_{percent}$. Then, in the destruction phase, d opened facilities are closed randomly by being removed from S. Afterward, the obtained partial solution is completed step by step using the greedy selection rule, and the feasible candidate solution is obtained again. The greediness of the selection is determined by parameter α, which can take values in $[0, 1]$ that 0 corresponds to a completely random selection whereas 1 corresponds to a completely greedy selection. Lastly, S is further tried to be improved by the local search algorithm and it is accepted as best solution so far if its objective value is greater than that of the S^*. If it is not accepted, S is restored with its previous value, which is S^*.

3.2. Greedy selection

A selection rule defines how to decide a new solution component that is going to be added for a partial solution, and used many times in construction and local search phases of the proposed algorithm. Adopted from Greedy Randomized Adaptive Search Procedure (Feo and Resende, 1995), the greedy selection rule that is used in this work is given in Alg. 3, and explained as follows. In the first step of the selection process, the candidate list (CL) is built by including the facilities that are not in S. Then, the facilities in CL are evaluated by $\Delta_{add}(\cdot)$ function that calculates the objective value change in case of a given facility is opened. Using the values of Δ_{min}, Δ_{max} and α, the restricted candidate list,

Algorithm 2: The proposed iterated greedy algorithm for solving OpM

```
input  : p, α, d_percent
output: S*
S ← GenerateSolutionRandomly();
S ← CompositeLocalSearch(S);
S* ← S ;
for i ← 1 to MAX_ITER do
    d ← p × d_percent;
    for i ← 1 to d do                                   ▷ Destruction phase
        k ← RandomSelection(S) ;
        S ← S \ {k};
    end
    for i ← 1 to d do                                   ▷ Construction phase
        l ← GreedySelection(S, α) ;
        S ← S ∪ {l};
    end
    S ← CompositeLocalSearch(S) ;                       ▷ Local search phase
    if f(S) > f(S*) then
        S* ← S;                    ▷ Accept S as the new best solution
    else
        S ← S*;                    ▷ Restore S with its previous value
    end
end
```

denoted by RCL, is constructed. In RCL construction, facilities in CL with higher Δ_{add} value are collected with respect to the parameter α. Finally, a random element is chosen from the RCL and returned as a selected facility. Note that the greediness of the selection is controlled by the parameter α. To be more precise, when it takes 0, all the CL elements are included in the RCL, hence a purely random selection is made. On the other hand, when it takes 1, only the first element of the CL which has the Δ_{max} value is included into RCL, hence a purely greedy selection is made. Generally, a good performing value for α is somewhere between 0 and 1, which is depending on a given problem instance and other parameters.

Algorithm 3: GreedySelection

input : S, α

output: l

1. CL $\leftarrow J \setminus S$;
2. $\Delta_{min} \leftarrow \min_{j \in CL} \Delta_{add}(j)$;
3. $\Delta_{max} \leftarrow \max_{j \in CL} \Delta_{add}(j)$;
4. RCL $\leftarrow \{j \in CL \mid \Delta_{add}(j) \geq \Delta_{min} + \alpha \times (\Delta_{max} - \Delta_{min})\}$;
5. $l \leftarrow$ RandomSelection(RCL) ;

3.3. Composite local search

Local search is an essential component for most of the metaheuristics because it contributes to exploitation behavior of the general search process. This study develops a composite local search (Alg. 4) that combines two low-level local search methods, namely RLS1 and RLS2, which were successfully used before to solve OpM by Herrán et al. (2018). The developed local search makes use of these two methods in a way that one is called after another as long as an improvement is obtained from one of the algorithms.

How RLS1 and RLS2 work is defined in Alg. 5 and Alg. 6, respectively, and explained as follows. Given $\Delta_{drop}(j) = f(S) - f(S \setminus \{j\})$ where, $j \in S$

Algorithm 4: CompositeLocalSearch

```
input  : S
output: S
improved ← true;
while improved do
    improved ← false;
    Δf ← RLS1(S);
    while Δf > 0 do
        improved ← true;
        Δf ← RLS1(S);
    end
    Δf ← RLS2(S);
    while Δf > 0 do
        improved ← true;
        Δf ← RLS2(S);
    end
end
```

and $\Delta_{add}(j) = f(S) - f(S \cup \{j\})$ where, $j \in J \setminus S$, RLS1 first removes a facility that has the maximum Δ_{drop} value and then adds a facility that has the maximum Δ_{add} value. On the other hand, RLS2 first adds a facility that has the maximum Δ_{add} value, and then removes a facility that has the maximum Δ_{drop} value. Although these two techniques appear to be similar, they can produce different neighborhoods, hence, result in different solutions.

Note that, $\Delta_{drop}(\cdot) \geq 0$ and $\Delta_{add}(\cdot) \leq 0$. So, if the absolute value of dropping gain is bigger that of adding loss, $\Delta f > 0$, and the solution is improved. Otherwise, in the worst case, the same facility is dropped and added, Δf gets zero, and the solution remains unchanged.

Algorithm 5: RLS1

input : S

output: $S, \Delta f$

$k \leftarrow \underset{j \in S}{\operatorname{argmax}} \Delta_{drop}(j)$;

$S \leftarrow S \setminus \{k\}$;

$l \leftarrow \underset{j \in J \setminus S}{\operatorname{argmax}} \Delta_{add}(j)$;

$S \leftarrow S \cup \{l\}$;

$\Delta f \leftarrow \Delta_{drop}(k) + \Delta_{add}(l)$;

Algorithm 6: RLS2

input : S

output: $S, \Delta f$

$l \leftarrow \underset{j \in J \setminus S}{\operatorname{argmax}} \Delta_{add}(j)$;

$S \leftarrow S \cup \{l\}$;

$k \leftarrow \underset{j \in S}{\operatorname{argmax}} \Delta_{drop}(j)$;

$S \leftarrow S \setminus \{k\}$;

$\Delta f \leftarrow \Delta_{drop}(k) + \Delta_{add}(l)$;

3.4. Solution structure and evaluation of the objective function

Most of the computation effort of the proposed algorithm is spent on destruction, construction and local search phases that are all based on adding or removing facilities to/from a solution at hand. So, it is important to use efficient methods to eliminate unnecessary calculations as much as possible. For this purpose, an auxiliary list (cf) that holds the closest facilities for each client as in (2) has been used.

$$\forall i \in I, cf_i = \underset{j \in S}{\operatorname{argmin}} \{d_{i,j}\} \tag{2}$$

Using cf, one can calculate the value of the objective function $f(\cdot)$ in $O(\mid I \mid)$ time as in (3).

$$f(S) = \sum_{i \in I} d_{i,cf_i} \tag{3}$$

Suppose that a facility $j \notin S$ is added to a solution S. Depending on the distance between the client i and the facility j, the new closet facilities list, denoted by cf', is either remains its previous value or takes j as in (4). Because both of the scenarios require $O(1)$ check operation per client, the overall time complexity for calculating the new closest facilities list is $O(\mid I \mid)$.

$$\forall i \in I, cf'_i = \begin{cases} cf_i, & \text{if } d_{i,j} > d_{i,cf_i} \\ j, & \text{otherwise} \end{cases} \tag{4}$$

On the other hand, suppose that a facility $j \in S$ is removed from a solution S. In the first case, the facility j is different than the cf_i, there will be no change. In the second case, the facility j is the same with the cf_i, so, there is a need to find the second minimum distant facility to replace the previous one. Considering these cases, the calculation of cf' after removing the facility j is done as in (5). Note that, for each client, the former case requires only $O(1)$ time; whereas the latter case requires $O(\mid S \mid)$ time since the linear search is performed on an unsorted list. In the best scenario in which the removed facility never exist in the cf, the overall time complexity will be $O(\mid I \mid)$. Contrarily, in the worst scenario in which all the values in cf equal the removed facility, the overall time complexity will be $O(\mid I \mid \times \mid S \mid)$.

$$\forall i \in I, cf'_i = \begin{cases} cf_i, & \text{if } j \neq cf_i \\ \operatorname*{argmin}_{j' \in S \setminus \{j\}} \{d_{i,j'}\}, & \text{otherwise} \end{cases} \tag{5}$$

4. Experimental work

The proposed IG algorithm was implemented in Visual C++ and ran on a computer with the configuration of Intel Core i7 6700, 3.40 GHz CPU using a single core.

Performance evaluation of the proposed algorithm has been carried out on OpM_LIB[1] benchmark instances. This benchmark consists of two instance lists, namely A and B. Described by Belotti et al. (2007), list A is generated by transforming 24 p-median instances (from pmed17 to pmed40) of OR-Library (Beasley, 1990) into 72 OpM instances. Then, list B is produced by transposing the matrix for each instance that includes distances between clients and facilities. Table 1 reports all the instance names and their properties, where n is the number of clients, m is the number of facilities and p is the number of facilities to be opened. Note that there exist A and B version for each instance, hence a total of 144 OpM instances are listed.

For the preliminary experiments, a total of 16 representative instances with different characteristics (marked as bold in Table 1) has been used instead of using the whole benchmark as suggested in (Herrán et al., 2018) in order to prevent the proposed algorithm from over-fitting.

It is also worth mentioning that the proposed algorithm has been run 50 times with different random seeds for all the experiments conducted in this paper due to the fact that IG is a probabilistic algorithm, and it may produce

[1] OpM_LIB benchmark is publicly available at http://grafo.etsii.urjc.es/optsicom/opm/.

Table 1: Instances generated from the OR-Library (Beasley, 1990)

Instance	n	m	p	Instance	n	m	p
pmed17-p100[A/B]	200	200	100	pmed29-p150[A/B]	300	300	150
pmed17-p25[A/B]	200	200	25	pmed29-p37[A/B]	300	300	37
pmed17-p50[A/B]	200	200	50	pmed29-p75[A/B]	300	300	75
pmed18-p100[A/B]	200	200	100	pmed30-p150[A/B]	300	300	150
pmed18-p25[A/B]	200	200	25	pmed30-p37[A/B]	300	300	37
pmed18-p50[A/B]	200	200	50	pmed30-p75[A/B]	300	300	75
pmed19-p100[A/B]	200	200	100	pmed31-p175[A/B]	350	350	175
pmed19-p25[A/B]	200	200	25	pmed31-p43[A/B]	350	350	43
pmed19-p50[A/B]	200	200	50	pmed31-p87[A/B]	350	350	87
pmed20-p100[A/B]	200	200	100	pmed32-p175[A/B]	350	350	175
pmed20-p25[A/B]	200	200	25	pmed32-p43[A/B]	350	350	43
pmed20-p50[A/B]	200	200	50	pmed32-p87[A/B]	350	350	87
pmed21-p125[A/B]	250	250	125	pmed33-p175[A/B]	350	350	175
pmed21-p31[A/B]	250	250	31	pmed33-p43[A/B]	350	350	43
pmed21-p62[A/B]	250	250	62	**pmed33-p87[A/B]**	350	350	87
pmed22-p125[A/B]	250	250	125	pmed34-p175[A/B]	350	350	175
pmed22-p31[A/B]	250	250	31	pmed34-p43[A/B]	350	350	43
pmed22-p62[A/B]	250	250	62	pmed34-p87[A/B]	350	350	87
pmed23-p125[A/B]	250	250	125	pmed35-p100[A/B]	400	400	100
pmed23-p31[A/B]	250	250	31	pmed35-p200[A/B]	400	400	200
pmed23-p62[A/B]	250	250	62	pmed35-p50[A/B]	400	400	50
pmed24-p125[A/B]	250	250	125	**pmed36-p100[A/B]**	400	400	100
pmed24-p31[A/B]	250	250	31	pmed36-p200[A/B]	400	400	200
pmed24-p62[A/B]	250	250	62	pmed36-p50[A/B]	400	400	50
pmed25-p125[A/B]	250	250	125	pmed37-p100[A/B]	400	400	100
pmed25-p31[A/B]	250	250	31	pmed37-p200[A/B]	400	400	200
pmed25-p62[A/B]	250	250	62	pmed37-p50[A/B]	400	400	50
pmed26-p150[A/B]	300	300	150	pmed38-p112[A/B]	450	450	112
pmed26-p37[A/B]	300	300	37	pmed38-p225[A/B]	450	450	225
pmed26-p75[A/B]	300	300	75	pmed38-p56[A/B]	450	450	56
pmed27-p150[A/B]	300	300	150	**pmed39-p112[A/B]**	450	450	112
pmed27-p37[A/B]	300	300	37	pmed39-p225[A/B]	450	450	225
pmed27-p75[A/B]	300	300	75	pmed39-p56[A/B]	450	450	56
pmed28-p150[A/B]	300	300	150	pmed40-p112[A/B]	450	450	112
pmed28-p37[A/B]	300	300	37	**pmed40-p225[A/B]**	450	450	225
pmed28-p75[A/B]	300	300	75	pmed40-p56[A/B]	450	450	56

different results for different runs.

4.1. Preliminary experiments

4.1.1. Parameter setting

Parameters affect the quality of produced solutions directly and it is important to find appropriate values to each of them. For this purpose, the irace package (López-Ibáñez et al., 2016), which is an automated parameter configuration tool based on iterative racing procedure, was used in this study to determine the values of the α and $d_{percent}$.

In the configuration of irace, the representative instances that belong to instance list A were selected as training instances whereas the representative instances that belong to instance list B were selected as test instances. The tuning budget was set to 1000 iterations, and the other settings were kept by default. In addition, MAX_ITER of the IG algorithm was set to $p \times 5$. The tuned values that were obtained after following this configuration can be seen in Table 2, and have been used in the rest of the computational study in this work.

Table 2: The tuned parameter values for the IG algorithm after using the irace

Param. name	Param. type	Tuning interval	Tuned values
α	real	[0.1, 0.9]	0.79
$d_{percent}$	real	[0.1, 0.9]	0.61

4.1.2. The effectiveness of the composite local search

This section analyzes the performance of the composite local search algorithm that is developed in this study. As explained before in Section 3.3, composite local search consists of RLS1 and RLS2 algorithms and uses them consecutively as long as an improvement is obtained. In order to measure how the developed local search contributes to the performance of the IG algorithm, these three cases have been considered: IG with RLS1, IG with RLS2 and IG with composite local search. To make a fair comparison, all the cases were run for the same amount of time budget of $p \times 0.01$ seconds.

Table 3: Impact of different local search strategies on the performance of the proposed algorithm. Cost, time (T.) and iteration (Iter.) values are averaged over 50 independent runs for each algorithm/instance pair.

	IG with RLS1			IG with RLS2			IG with Composite LS		
Instance	Cost	T. (s)	Iter.	Cost	T. (s)	Iter.	Cost	T. (s)	Iter.
pmed17-p25A	6261.82±87.26	0.25	148.1	6261.50±101.15	0.25	145.82	7317.00±0.00	0.25	56.14
pmed20-p50A	5307.82±57.82	0.50	250.2	5302.44±62.77	0.50	244.24	5871.82±1.27	0.50	122.82
pmed22-p62A	5146.06±73.11	0.62	169.5	5146.34±76.75	0.62	170.76	5992.54±5.42	0.62	78.86
pmed28-p75A	4496.32±71.02	0.75	121.0	4505.42±77.66	0.75	119.16	5670.16±6.74	0.76	50.2
pmed33-p87A	4943.32±60.88	0.88	87.1	4943.52±58.16	0.87	84.86	5784.14±10.82	0.88	41.68
pmed36-p100A	5266.42±78.43	1.01	62.8	5262.94±73.61	1.01	62.76	6453.90±6.33	1.02	30.64
pmed39-p112A	4755.30±70.11	1.13	53.0	4750.54±62.40	1.13	52.18	5927.96±9.57	1.14	24.04
pmed40-p225A	4036.60±56.32	2.27	68.9	4034.50±57.04	2.27	69.18	4560.48±5.39	2.28	42.16
pmed17-p25B	6124.56±76.50	0.25	154.4	6121.00±83.79	0.25	152.56	6905.00±0.00	0.25	62.18
pmed20-p50B	4901.84±71.02	0.50	272.6	4899.00±73.51	0.50	270.6	5665.00±0.00	0.50	120.3
pmed22-p62B	5075.12±56.91	0.62	161.5	5081.90±47.81	0.62	164.24	6259.00±0.00	0.62	69.5
pmed28-p75B	4714.18±48.72	0.75	114.8	4721.88±52.41	0.75	115.92	5625.08±7.19	0.76	53.42
pmed33-p87B	4925.46±58.38	0.88	81.6	4937.18±60.98	0.88	82.1	5823.44±9.34	0.88	41.06
pmed36-p100B	5172.10±63.70	1.01	63.5	5168.04±55.32	1.01	63.84	6193.76±18.48	1.02	31.76
pmed39-p112B	4691.42±76.08	1.13	52.1	4688.82±76.40	1.13	51.7	6183.80±10.32	1.15	23.68
pmed40-p225B	4183.70±49.93	2.27	68.4	4188.42±51.93	2.27	68.68	4512.92±5.54	2.27	44.26
Avg.	5000.13±66.01	0.93	120.60	5000.84±66.98	0.93	119.91	5921.63±6.03	0.93	55.79
Wilcox. S.R.	$p < 0.001$			$p < 0.001$					

The results obtained for 16 representative instances are listed in Table 3. By averaging over 50 runs, the column "Cost" reports the maximized objective function value, the column "T." reports the elapsed CPU time in seconds, and the column "Iter." reports the iteration count when the algorithm terminates. Average results show that IG with composite local search reaches the lowest average iteration count in a given time budget since it requires more CPU time than both RLS1 and RLS2. However, it is seen that the average cost value of the composite local search is overwhelmingly better than those of both RLS1 and RLS2. Also, the lower standard deviation values show the robustness of the composite local search in a given limited time. The difference between the developed composite local search and the two others has also been tested by Wilcoxon signed-rank method which is a non-parametric statistical test to compare two related samples. The obtained $p < 0.001$ values indicate that these differences are both statistically significant for a selected representative instance set.

4.1.3. Computational results over the whole set of instances

In this section, the performance of the proposed algorithm is evaluated over the whole set of instances provided in OpM_LIB benchmark. After some preliminary testing, the termination condition of the algorithm, MAX_ITER, is set to $p \times 10$ for each problem instance.

The obtained computational results are presented in Table 4 and Table 5 for instance lists A and B, respectively. BKS denotes the cost value of a best-known solution for each instance, taken from Herrán et al. (2018), Lin and Guan (2018) and Mladenovic et al. (2019). "Best" and "Avg." columns give the best and average cost values obtained from the algorithm after 50 runs, respectively. The column "Dev." lists the deviation of the average cost in percentage with respect to the BKS values for each instance i, calculated as $\frac{BKS_i - Cost_i}{BKS_i} \times 100$. The column "Succ." gives how many times the algorithm reaches or exceeds the BKS value. The column "CV" corresponds to the coefficient of variation and presents the relative standard deviation for each instance, calculated as $\frac{StandardDeviation_i}{Mean_i} \times 100$. The column "#Eval." provides the average number of objective value change evaluations (including opening or closing calculations) required to reach the final solution per instance. Finally, the column "T.(s)" lists the average CPU times in seconds that were spent by the algorithm.

Table 4 reports the computational results of the algorithm for instance set A. It is seen that the proposed algorithm has reached BKS value for all the instances in terms of best cost values. In terms of average cost, the algorithm can achieve BKS values in 47 out of 72 instances. It is also seen that the average success rate is approximately 43.13/50 and the average CV value is smaller than 0.02, which reveals the robustness of the proposed algorithm. As another important performance metric, the algorithm can achieve approximately 22 seconds of CPU time on average.

Similarly, Table 5 reports the computational results of the algorithm for instance set B. It is seen that the general performance of the algorithm over this set is akin to that of set A. More specifically, the algorithm has reached BKS

Table 4: Computational results for the instances in set A: boldface indicates that the cost of a BKS is reached; * indicates that the cost of a BKS is improved.

Instance	BKS	Best	Avg.	Dev.	Succ.	CV	#Eval.	T. (s)
pmed17-p100A	4054	**4054**	**4054.00**	0.000	50	0.000	76492.72	4.58
pmed17-p25A	7317	**7317**	**7317.00**	0.000	50	0.000	20843.02	1.21
pmed17-p50A	5411	**5411**	**5411.00**	0.000	50	0.000	83391.44	2.28
pmed18-p100A	4220	**4220**	**4220.00**	0.000	50	0.000	103398.94	4.29
pmed18-p25A	7432	**7432**	**7432.00**	0.000	50	0.000	16460.56	1.10
pmed18-p50A	5746	**5746**	**5746.00**	0.000	50	0.000	50996.50	2.23
pmed19-p100A	4033	**4033**	**4033.00**	0.000	50	0.000	119121.52	5.01
pmed19-p25A	7020	**7020**	**7020.00**	0.000	50	0.000	13249.42	1.18
pmed19-p50A	5387	**5387**	5386.34	0.012	17	0.009	81452.96	2.20
pmed20-p100A	4063	**4063**	**4063.00**	0.000	50	0.000	98296.78	4.50
pmed20-p25A	7648	**7648**	**7648.00**	0.000	50	0.000	14289.88	1.16
pmed20-p50A	5872	**5872**	**5872.00**	0.000	50	0.000	56764.40	2.28
pmed21-p125A	4155	**4155**	4154.96	0.001	49	0.007	171923.60	12.49
pmed21-p31A	7304	**7304**	**7304.00**	0.000	50	0.000	41267.30	2.40
pmed21-p62A	5784	**5784**	5782.98	0.018	33	0.054	156273.30	5.57
pmed22-p125A	4358	**4358**	4353.82	0.096	24	0.097	194032.24	10.28
pmed22-p31A	7900	**7900**	**7900.00**	0.000	50	0.000	45491.78	2.49
pmed22-p62A	5995	**5995**	**5995.00**	0.000	50	0.000	103239.70	5.09
pmed23-p125A	4114	**4114**	**4114.00**	0.000	50	0.000	251046.64	11.85
pmed23-p31A	7841	**7841**	**7841.00**	0.000	50	0.000	20439.26	2.70
pmed23-p62A	5785	**5785**	**5785.00**	0.000	50	0.000	125620.42	5.61
pmed24-p125A	4091	**4091**	**4091.00**	0.000	50	0.000	206143.08	13.52
pmed24-p31A	7425	**7425**	**7425.00**	0.000	50	0.000	24486.12	2.41
pmed24-p62A	5528	**5528**	**5528.00**	0.000	50	0.000	106321.50	5.04
pmed25-p125A	4155	**4155**	4154.78	0.005	46	0.023	206411.28	13.19
pmed25-p31A	7552	**7552**	**7552.00**	0.000	50	0.000	19594.68	2.52
pmed25-p62A	5767	**5767**	**5767.00**	0.000	50	0.000	125726.40	5.88
pmed26-p150A	4341	**4341**	4340.30	0.016	35	0.026	323149.60	24.76
pmed26-p37A	8112	**8112**	**8112.00**	0.000	50	0.000	11636.98	5.17
pmed26-p75A	5789	**5789**	**5789.00**	0.000	50	0.000	192238.28	11.14
pmed27-p150A	4062	**4062**	4061.94	0.001	49	0.010	338674.76	25.51
pmed27-p37A	7556	**7556**	**7556.00**	0.000	50	0.000	61133.62	5.07
pmed27-p75A	5668	**5668**	5667.08	0.016	43	0.043	210397.54	11.23
pmed28-p150A	4099	**4099**	**4099.00**	0.000	50	0.000	282403.30	21.22
pmed28-p37A	7366	**7366**	**7366.00**	0.000	50	0.000	55894.68	5.10
pmed28-p75A	5681	**5681**	**5681.00**	0.000	50	0.000	184276.72	11.57
pmed29-p150A	4141	**4141**	4139.76	0.030	15	0.031	349982.56	25.91
pmed29-p37A	7404	**7404**	**7404.00**	0.000	50	0.000	73068.18	4.63
pmed29-p75A	5880	**5880**	**5880.00**	0.000	50	0.000	144484.92	11.23
pmed30-p150A	4385	**4385**	**4385.00**	0.000	50	0.000	265179.24	22.75
pmed30-p37A	7704	**7704**	**7704.00**	0.000	50	0.000	51690.10	4.50
pmed30-p75A	6189	**6189**	6186.50	0.040	25	0.041	195791.06	11.31
pmed31-p175A	4136	**4136**	4134.80	0.029	3	0.014	482716.36	48.94
pmed31-p43A	7424	**7424**	**7424.00**	0.000	50	0.000	86526.98	7.99
pmed31-p87A	5905	**5905**	**5905.00**	0.000	50	0.000	200912.32	20.23
pmed32-p175A	4242	**4242**	4241.62	0.009	38	0.017	399891.16	41.47
pmed32-p43A	7794	**7794**	**7794.00**	0.000	50	0.000	99056.42	8.17
pmed32-p87A	5925	**5925**	5924.60	0.007	49	0.048	282371.30	19.21
pmed33-p175A	4105	**4105**	4102.30	0.066	2	0.031	443629.54	42.85
pmed33-p43A	7598	**7598**	**7598.00**	0.000	50	0.000	89898.76	7.93
pmed33-p87A	5793	**5793**	**5793.00**	0.000	50	0.000	275727.56	18.65
pmed34-p175A	4287	**4287**	**4287.00**	0.000	50	0.000	438460.12	44.15
pmed34-p43A	7725	**7725**	**7725.00**	0.000	50	0.000	108860.76	8.12
pmed34-p87A	5849	**5849**	5847.08	0.033	31	0.042	262842.74	19.47
pmed35-p100A	5845	**5845**	5844.96	0.001	48	0.003	368679.06	34.67
pmed35-p200A	4007	**4007**	4005.36	0.041	15	0.034	656431.70	79.02
pmed35-p50A	7155	**7155**	**7155.00**	0.000	50	0.000	141758.88	13.36
pmed36-p100A	6461	**6461**	**6461.00**	0.000	50	0.000	269865.26	33.08
pmed36-p200A	4319	**4319**	4317.46	0.036	29	0.073	647564.28	73.98
pmed36-p50A	8179	**8179**	**8179.00**	0.000	50	0.000	125740.54	13.38
pmed37-p100A	6203	**6203**	6202.44	0.009	40	0.022	394641.12	31.54
pmed37-p200A	4593	**4593**	4590.42	0.056	22	0.063	688083.06	77.03
pmed37-p50A	7830	**7830**	**7830.00**	0.000	50	0.000	172276.30	11.73
pmed38-p112A	5915	**5915**	5914.42	0.010	39	0.022	470285.26	52.80
pmed38-p225A	4428	**4428**	4426.74	0.028	20	0.024	859027.60	129.24
pmed38-p56A	7432	**7432**	**7432.00**	0.000	50	0.000	141119.52	19.48
pmed39-p112A	5935	**5935**	**5935.00**	0.000	50	0.000	406560.94	52.29
pmed39-p225A	4369	**4369**	4368.62	0.009	31	0.011	819629.72	124.18
pmed39-p56A	7712	**7712**	**7712.00**	0.000	50	0.000	146126.50	20.75
pmed40-p112A	6272	**6272**	6271.90	0.002	45	0.005	445914.72	49.43
pmed40-p225A	4572	**4572**	4570.66	0.029	7	0.021	857585.62	124.02
pmed40-p56A	8211	**8211**	**8211.00**	0.000	50	0.000	173055.64	19.70
Avg.	5896.60	5896.60	5896.21	0.008	43.13	0.011	225389.12	21.96

Table 5: Computational results for the instances in set B: boldface indicates that the cost of a BKS is reached; * indicates that the cost of a BKS is improved.

Instance	BKS	Best	Avg.	Dev.	Succ.	CV	# Eval.	T. (s)
pmed17-p100B	3992	**3992**	**3992.00**	0.000	50	0.000	75947.28	5.41
pmed17-p25B	6905	**6905**	**6905.00**	0.000	50	0.000	17580.46	1.11
pmed17-p50B	5563	**5563**	**5563.00**	0.000	50	0.000	73173.38	2.66
pmed18-p100B	4122	**4122**	4121.52	0.012	42	0.027	114919.06	4.29
pmed18-p25B	7662	**7662**	**7662.00**	0.000	50	0.000	24361.32	1.14
pmed18-p50B	5852	**5852**	**5852.00**	0.000	50	0.000	59247.56	2.28
pmed19-p100B	4016	**4016**	**4016.00**	0.000	50	0.000	95052.58	4.62
pmed19-p25B	6816	**6816**	**6816.00**	0.000	50	0.000	13700.66	1.05
pmed19-p50B	5423	**5423**	**5423.00**	0.000	50	0.000	64291.10	2.37
pmed20-p100B	4067	**4067**	**4067.00**	0.000	50	0.000	135298.72	4.60
pmed20-p25B	7349	**7349**	**7349.00**	0.000	50	0.000	12944.18	1.12
pmed20-p50B	5665	**5665**	**5665.00**	0.000	50	0.000	51935.52	2.26
pmed21-p125B	4033	**4033**	4032.72	0.007	47	0.036	212941.32	11.73
pmed21-p31B	7331	**7331**	**7331.00**	0.000	50	0.000	29192.56	2.61
pmed21-p62B	5870	**5870**	**5870.00**	0.000	50	0.000	86841.44	5.75
pmed22-p125B	4338	**4338**	4336.88	0.026	23	0.026	243202.30	11.79
pmed22-p31B	7695	**7695**	**7695.00**	0.000	50	0.000	22269.88	2.55
pmed22-p62B	6259	**6259**	**6259.00**	0.000	50	0.000	76628.88	6.05
pmed23-p125B	4095	**4095**	**4095.00**	0.000	50	0.000	189044.66	11.22
pmed23-p31B	7137	**7137**	**7137.00**	0.000	50	0.000	47420.34	2.37
pmed23-p62B	5724	**5724**	**5724.00**	0.000	50	0.000	101162.88	5.27
pmed24-p125B	4072	**4072**	**4072.00**	0.000	50	0.000	222665.58	12.54
pmed24-p31B	7190	**7190**	**7190.00**	0.000	50	0.000	40677.54	2.28
pmed24-p62B	5752	**5752**	5750.54	0.025	47	0.102	129046.74	5.43
pmed25-p125B	4233	**4233**	4230.84	0.051	29	0.063	181880.46	11.48
pmed25-p31B	7552	**7552**	**7552.00**	0.000	50	0.000	51615.12	2.66
pmed25-p62B	5692	**5692**	5691.80	0.004	49	0.025	133719.30	5.91
pmed26-p150B	4173	**4173**	**4173.00**	0.000	50	0.000	347892.78	26.28
pmed26-p37B	7643	**7643**	**7643.00**	0.000	50	0.000	50942.76	4.86
pmed26-p75B	5923	**5923**	**5923.00**	0.000	50	0.000	157197.62	11.72
pmed27-p150B	4144	**4144**	4143.92	0.002	49	0.014	314088.52	26.25
pmed27-p37B	7448	**7448**	**7448.00**	0.000	50	0.000	54127.20	5.05
pmed27-p75B	5844	**5844**	**5844.00**	0.000	50	0.000	190412.72	12.64
pmed28-p150B	4069	**4069**	4068.88	0.003	44	0.008	339728.82	25.65
pmed28-p37B	7388	**7388**	**7388.00**	0.000	50	0.000	44762.26	4.99
pmed28-p75B	5642	**5642**	5639.88	0.038	36	0.066	216554.46	11.46
pmed29-p150B	4157	**4157**	**4157.00**	0.000	50	0.000	300338.42	23.76
pmed29-p37B	7529	**7529**	**7529.00**	0.000	50	0.000	53743.08	4.96
pmed29-p75B	5709	**5709**	**5709.00**	0.000	50	0.000	204382.20	11.41
pmed30-p150B	4313	**4313**	4312.84	0.004	47	0.016	377042.88	25.69
pmed30-p37B	8048	**8048**	**8048.00**	0.000	50	0.000	37828.80	4.72
pmed30-p75B	6041	**6041**	**6041.00**	0.000	50	0.000	185069.42	10.60
pmed31-p175B	4138	**4138**	4137.64	0.009	49	0.062	448719.04	44.46
pmed31-p43B	7320	**7320**	**7320.00**	0.000	50	0.000	101406.82	8.15
pmed31-p87B	5621	**5621**	5617.52	0.062	19	0.057	312222.62	19.87
pmed32-p175B	4244	**4247***	4242.00	0.047	44	0.185	435546.82	43.00
pmed32-p43B	7899	**7899**	**7899.00**	0.000	50	0.000	79251.30	7.94
pmed32-p87B	5852	**5852**	5845.64	0.109	16	0.082	317744.18	18.89
pmed33-p175B	4156	**4156**	4154.72	0.031	35	0.053	475575.40	44.39
pmed33-p43B	7611	**7611**	**7611.00**	0.000	50	0.000	113690.78	7.48
pmed33-p87B	5840	**5840**	5838.98	0.017	33	0.028	321927.26	18.88
pmed34-p175B	4270	**4270**	**4270.00**	0.000	50	0.000	417589.12	47.33
pmed34-p43B	7514	**7514**	**7514.00**	0.000	50	0.000	72416.52	8.18
pmed34-p87B	5857	**5857**	5855.92	0.018	29	0.023	309946.88	19.34
pmed35-p100B	5639	**5639**	**5639.00**	0.000	50	0.000	349795.94	31.02
pmed35-p200B	4109	**4109**	4108.36	0.016	27	0.022	671362.92	76.70
pmed35-p50B	7570	**7570**	**7570.00**	0.000	50	0.000	103358.34	14.41
pmed36-p100B	6219	**6219**	6215.80	0.051	25	0.055	417711.82	31.90
pmed36-p200B	4319	**4321***	4318.46	0.013	31	0.036	613311.64	67.68
pmed36-p50B	8144	**8144**	**8144.00**	0.000	50	0.000	125580.14	13.30
pmed37-p100B	6211	**6212***	**6209.16**	0.030	6	0.032	417010.04	30.90
pmed37-p200B	4609	**4609**	4608.60	0.009	40	0.018	621983.56	81.69
pmed37-p50B	8379	**8379**	**8379.00**	0.000	50	0.000	91141.04	12.57
pmed38-p112B	5949	**5949**	5948.56	0.007	39	0.014	537907.22	52.92
pmed38-p225B	4446	**4446**	4443.46	0.057	23	0.073	834419.68	136.40
pmed38-p56B	7535	**7535**	**7535.00**	0.000	50	0.000	171157.68	20.89
pmed39-p112B	6198	**6198**	**6198.00**	0.000	50	0.000	450664.08	53.22
pmed39-p225B	4266	**4267***	4264.04	0.046	11	0.052	763636.10	125.43
pmed39-p56B	7625	**7625**	**7625.00**	0.000	50	0.000	181289.80	20.67
pmed40-p112B	6200	**6200**	6199.68	0.005	38	0.011	557014.60	49.06
pmed40-p225B	4525	**4532***	**4529.82**	-0.107	44	0.073	904935.12	115.71
pmed40-p56B	8022	**8022**	**8022.00**	0.000	50	0.000	192937.04	19.22
Avg.	5871.71	5871.90	5871.28	0.008	44.06	0.017	233223.98	22.00

value for all the instances in terms of best costs. Also, the proposed algorithm could produce new best solutions for the 5 instances, namely pmed32-p175B, pmed36-p200B, pmed37-p100B, pmed39-p225B, and pmed40-p225B. Furthermore, in terms of average cost, it can achieve BKS values in 46 out of 72 instances. In addition, the average success rate is approximately 44.06/50, the average CV value is smaller than 0.02, and the average CPU time is around 22 seconds. Overall, it can be concluded that the proposed IG algorithm can produce high-quality solutions for OpM problems in a reasonable time.

4.2. Comparison with state-of-the-art

In this section, the proposed IG algorithm is compared with state-of-the-art algorithms for the OpM. For this purpose, the following algorithms in literature have been selected: standard Branch and Cut (BC) and Tabu Search based BC (XTS) from Belotti et al. (2007); Greedy Randomized Adaptive Search Procedure (GRASP) from Colmenar et al. (2016); Hybrid Binary Particle Swarm Optimization (HBPSO) from Lin and Guan (2018), parallel Variable Neighborhood Search (P-VNS) from Herrán et al. (2018) and basic Variable Neighborhood Search (VNS) from Mladenovic et al. (2019).

The computational results of the algorithms BC, XTS, GRASP, and P-VNS are presented in Herrán et al. (2018) for both of the instance sets A and B, and produced by the same computer with the configuration of Intel Core i5 660, 3.3 GHz. Because average and best costs are not explicitly stated in computational results for these algorithms, it is assumed that the reported results are based on a single run. On the other hand, the results of VNS have been presented in both best and average costs (for 30 runs), produced by the computer with the configuration of Intel Xeon E7 4820 CPU, 2.00 GHz. As for HBPSO, the results have only been presented for instance set A, reported in both best and average costs (for 40 runs), and produced by the computer with the configuration of AMD A4-5300, 3.4 GHz. To make a fair comparison of running times between the algorithms, the reported CPU times have been normalized according to their single thread performance scores that are obtained from https://www.cpubenchmark.net.

Comparison of the proposed IG algorithm with other algorithms over the combined instance sets of A and B is given in Table 6. For HBPSO, computational results are only available for instance set A, therefore, a second comparison that has been made with this algorithm is given in Table 7. It can be seen from the tables that the proposed algorithm outperforms BC, XTS, GRASP, and VNS implementations in terms of average cost, best cost (if available) and running time performances. Although average CPU times of HBPSO and IG are close with each other, average and best cost performances of the IG is better. P-VNS is the only algorithm that produces better average cost (5884.0) than that of the proposed (5883.7), but this difference is so small when the average CPU times of both algorithms are considered (31.5 vs. 21.98). Moreover, compared to P-VNS, which has a parallel search mechanism that requires the design of solution exchange strategies, the proposed IG algorithm has a simpler algorithmic structure with ease of implementation.

Table 6: Comparison of the proposed IG algorithm with other algorithms over the combined instance sets of A and B.

	BC	XTS	GRASP	P-VNS	VNS	IG
Avg. Cost	5775.7	5815.8	5881.0	5884.0	5878.3	5883.7
Best Cost	N/A	N/A	N/A	N/A	5884.0	5884.3
Time (sec.)	5338.9	333.8	432.0	48.7	199.3	21.98
CPU score	1.394	1.394	1.394	1.394	1.162	2.155
Scale	0.647	0.647	0.647	0.647	0.539	1.000
Time (Scaled)	3453.6	215.9	279.4	31.5	107.5	21.98

Table 7: Comparison of the proposed IG algorithm with HBPSO over the instance set A.

	HBPSO	IG
Avg. Cost	5894.7	5896.2
Best Cost	5896.5	5896.6
Time (sec.)	38.9	21.96
CPU score	1.241	2.155
Scale	0.576	1.000
Time (Scaled)	22.4	21.96

5. Conclusion

In this study, an optimization algorithm based on Iterated Greedy (IG) metaheuristic has been proposed to solve the obnoxious p-median problem (OpM). In the construction phase of the IG algorithm Greedy Randomized Adaptive Search Procedure based selection criterion has been used. In addition, a composite local search method has been developed using RLS1 and RLS2, which were individually and successfully applied to solve the OpM before. The performance of the proposed algorithm was tested on a common benchmark consisting of 144 problem instances.

Experimental work shows that the proposed IG algorithm is highly effective for solving the OpM. The results indicate that, based on the set of selected instances, the proposed method outperforms most of the state-of-the-art counterparts including XTS, GRASP, VNS, and HBPSO implementations in terms of both average cost and running time. While P-VNS is the only method that exceeds the average cost performance of the developed IG algorithm, the cost difference between the two algorithms is very small and the proposed algorithm works much faster.

Future research might concentrate on the application of adaptive parameter control techniques to the IG algorithm so that the algorithm adapts itself better for each problem instance. Moreover, the running time of the algorithm can be further decreased by the parallel evaluation of multiple solution candidates.

Funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Conflicts of interest

Declarations of interest: none

References

Arroyo, J.E.C., Leung, J.Y.T., Tavares, R.G., 2019. An iterated greedy algorithm for total flow time minimization in unrelated parallel batch machines with unequal job release times. Eng. Appl. of Artif. Intell. 77, 239–254.

Beasley, J.E., 1990. OR-Library: Distributing Test Problems by Electronic Mail. J. of the Oper. Res. Soc. 41, 1069–1072. URL: https://www.tandfonline.com/doi/full/10.1057/jors.1990.166, doi:10.1057/jors.1990.166.

Belotti, P., Labbé, M., Maffioli, F., Ndiaye, M.M., 2007. A branch-and-cut method for the obnoxious p-median problem. 4OR 5, 299–314. URL: http://link.springer.com/10.1007/s10288-006-0023-3, doi:10.1007/s10288-006-0023-3.

Bouamama, S., Blum, C., Boukerram, A., 2012. A population-based iterated greedy algorithm for the minimum weight vertex cover problem. Appl. Soft Comput. 12, 1632–1639.

Church, R.L., Garfinkel, R.S., 1978. Locating an obnoxious facility on a network. Transp. sci. 12, 107–118.

Colmenar, J.M., Greistorfer, P., Martí, R., Duarte, A., 2016. Advanced Greedy Randomized Adaptive Search Procedure for the Obnoxious p-Median problem. Eur. J. of Oper. Res. doi:10.1016/j.ejor.2016.01.047.

Current, J., Daskin, M., Schilling, D., 2002. Discrete network location models, in: Drezner, Zvi, H.H.W. (Ed.), Facility location: Applications and theory. Springer-Verlag Berlin Heidelberg, pp. 81–118.

Dell'Amico, M., Lodi, A., Maffioli, F., 1999. Solution of the cumulative assignment problem with a well-structured tabu search method. J. of Heuristics 5, 123–143.

Eberhart, R., Kennedy, J., 1995. A new optimizer using particle swarm theory, in: MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Ieee. pp. 39–43.

Farahani, R.Z., Hekmatfar, M., 2009. Facility location: concepts, models, algorithms and case studies. Springer.

Feo, T.A., Resende, M.G.C., 1995. Greedy Randomized Adaptive Search Procedures. J. of Glob. Optim. 6, 109–133. URL: http://link.springer.com/10.1007/BF01096763, doi:10.1007/BF01096763.

García-Martínez, C., Rodriguez, F.J., Lozano, M., 2014. Tabu-enhanced iterated greedy algorithm: a case study in the quadratic multiple knapsack problem. Eur. J. of Oper. Res. 232, 454–463.

Herrán, A., Colmenar, J.M., Martí, R., Duarte, A., 2018. A parallel variable neighborhood search approach for the obnoxious p-median problem. Int. Trans. in Oper. Res. URL: http://doi.wiley.com/10.1111/itor.12510, doi:10.1111/itor.12510.

Karabulut, K., Tasgetiren, M.F., 2014. A variable iterated greedy algorithm for the traveling salesman problem with time windows. Inf. Sci. 279, 383–395.

Lin, G., Guan, J., 2018. A hybrid binary particle swarm optimization for the obnoxious p-median problem. Inf. Sci. 425, 1–17.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives 3, 43–58.

Lourenço, H.R., Martin, O.C., Stützle, T., 2003. Iterated local search, in: Handbook of metaheuristics. Springer, pp. 320–353.

Mitchell, J.E., 2002. Branch-and-cut algorithms for combinatorial optimization problems. Handb. of appl. optim. 1, 65–77.

Mladenovic, N., Alkandari, A., Pei, J., Todosijevi, R., Pardalos, P.M., 2019. Less is more approach: basic variable neighborhood search for the obnoxious p-median problem. Int. Trans. in Oper. Res. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12646, doi:10.1111/itor.12646.

Mladenović, N., Hansen, P., 1997. Variable neighborhood search. Comput. & oper. res. 24, 1097–1100.

Nucamendi-Guillén, S., Angel-Bello, F., Martínez-Salazar, I., Cordero-Franco, A.E., 2018. The cumulative capacitated vehicle routing problem: New formulations and iterated greedy algorithms. Expert Syst. with Appl. 113, 315–327.

Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. of Oper. Res. 177, 2033–2049. URL: https://www.sciencedirect.com/science/article/pii/S0377221705008507, doi:10.1016/J.EJOR.2005.12.009.

Stützle, T., Ruiz, R., 2018. Iterated Greedy. Springer International Publishing, Cham. pp. 547–577. URL: https://doi.org/10.1007/978-3-319-07124-4_10, doi:10.1007/978-3-319-07124-4_10.

Tamir, A., 1991. Obnoxious Facility Location on Graphs. SIAM J. on Discret. Math. 4, 550–567. URL: http://epubs.siam.org/doi/10.1137/0404048, doi:10.1137/0404048.

Wolpert, D.H., Macready, W.G., et al., 1997. No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1, 67–82.