INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 0 (2023) 1–32
DOI: 10.1111/itor.13340

An iterated greedy algorithm with variable reconstruction size for the obnoxious *p*-median problem

Seyed Mousavi[a,*], Soni Bhambar[b] and Matthew England[b]

[a] *School of Computing, Mathematics and Data Sciences, Coventry University, Coventry CV1 5FB, UK*
[b] *Centre for Computational Science and Mathematical Modelling, Coventry University, Coventry CV1 5FB, UK*
E-mail: Seyed.Mousavi@coventry.ac.uk [Mousavi]; bhambars@uni.coventry.ac.uk [Bhambar]; Matthew.England@coventry.ac.uk [England]

Received 19 December 2022; received in revised form 15 May 2023; accepted 31 May 2023

Abstract

The obnoxious p-median problem is a facility location problem where we maximise the sum of the distances between each client point and its nearest facility. Since it is nondeterministic polynomial-time (NP)-hard, most algorithms designed for the problem follow metaheuristic strategies to find high-quality solutions in affordable time but with no optimality guarantee. In this paper, a variant of the iterated greedy algorithm is developed for the problem. It adopts the idea of increasing the search radius used in variable neighbourhood search by increasing the number of reconstructed components at each iteration with no improved solution, where the amount of the increase is determined dynamically based on the quality of the current solution. We demonstrate that the new algorithm significantly outperforms the current state-of-the-art metaheuristic algorithms for this problem on standard datasets.

Keywords: facility location; hybrid metaheuristic; iterated greedy; obnoxious p-median problem; p-median problem

1. Introduction

The *obnoxious p-median* (OpM) problem is to locate a given number of facilities such that the sum of the distances between each client point and its nearest facility is maximised. It is similar to the *p-median* (pM) problem where we instead minimise that sum. The OpM problem has numerous real-world applications where the facilities, although useful, are unpleasant (hence obnoxious) to nearby clients. Examples of such facilities include garbage collection points in residential areas and the positioning of airports (obnoxious due to their air and sound pollution). Another example, by Chang et al. (2021), is the location of quarantine sites during pandemics. These sites should be far from residential areas to reduce the chance of infections. Among other applications of this

*Corresponding author.

problem are the location of chemical and nuclear sites (Gökalp, 2020), hazardous waste disposal facilities, and high-voltage electrical transmission stations (Church and Drezner, 2022). Therefore, solving the OpM problem can potentially benefit people in a wide variety of ways. In addition, any algorithm for this problem can be a potential algorithm, via minor adjustments, for the pM problem because of their similarity.

1.1. Prior work

Research on this problem and its variants began in the last century when optimal solutions were sought (Church and Garfinkel, 1978; Erkut and Neuman, 1989; Plastria, 1996). However, the problem is NP-hard (Tamir, 1991), which means there exists no polynomial-time exact algorithm unless $P = NP$. Therefore, more recent research has focused on inexact algorithms, in particular those following metaheuristic strategies, to find (high-quality) feasible solutions in affordable time but with no optimality guarantee (Belotti et al., 2007; Colmenar et al., 2016a; Lin and Guan, 2018; Gökalp, 2020; Herrán et al., 2020; Mladenović et al., 2020).

Belotti et al. (2007) proposed a variant of tabu search (TS) called eXploring TS (X-TS), in addition to an exact branch and cut algorithm. Colmenar et al. (2016a) improved on the X-TS algorithm by proposing a Greedy Randomised Adaptive Search Procedure (GRASP) equipped with further mechanisms. For example, they used a filtering method to avoid local search on low-quality solutions. They also maintained for each client two sorted lists of open and closed facilities to speed up their local search. Herrán et al. (2020) proposed an improved algorithm together with its parallel version based on variable neighbourhood search (VNS). Among useful ideas in their work was to decouple the facility swap operation used in Colmenar et al. (2016a) into two single operations of dropping and adding a facility. Then, by using different orders of these two operations, they obtained two different local search procedures. The decoupling idea was also used in Lin and Guan (2018), where a hybrid of binary particle swarm optimisation and iterated greedy (IG) algorithms was proposed to improve on the GRASP algorithm of Colmenar et al. (2016a). Mladenović et al. (2020) also used a basic VNS algorithm based on the so-called less-is-more-approach (Mladenović et al., 2016). They showed that the resulting algorithm was superior to the GRASP algorithm of Colmenar et al. (2016a) and competitive with the VNS metaheuristic algorithm of Herrán et al. (2020).

Gökalp (2020) extended the IG algorithm presented by Lin and Guan (2018) and further enhanced it by applying the local search procedures proposed by Herrán et al. (2020). He showed his algorithm outperformed the VNS algorithm of Herrán et al. (2020). Most recently, another TS method was proposed by Chang et al. (2021). It was not compared with the IG algorithm of Gökalp (2020) but was shown to outperform the other state-of-the-art metaheuristic algorithms at the time.

Thus, to the best of our knowledge, the current state-of-the-art metaheuristic algorithms for the problem are the IG of Gökalp (2020) and the TS of Chang et al. (2021).

For a review of the existing models for the problem and its historical overview, the interested reader is referred to Church and Drezner (2022).

1.2. Our contribution

The algorithm proposed in the present paper is a variant of IG with variable reconstruction size. It incorporates the idea of increasing the neighbourhood radius (for diversification) used in VNS by increasing the number of reconstructed components. However, in contrast to the standard VNS approach where the amount of the increase is fixed, in our proposed algorithm, it is variable and determined dynamically at run time. The new algorithm also generalises the idea of applying an additional pair of construction and deconstruction operations used in Herrán et al. (2020) and Gökalp (2020) to improve the solution quality. More specifically, in contrast to Herrán et al. (2020) and Gökalp (2020), its construction and deconstruction operators are not limited to a local search stage or a fixed number of components. In addition, it uses two data structures not previously proposed in the literature of OpM, to the best of our knowledge.

Although it hybridises several ideas and operations, the overall algorithm is actually simpler than the state-of-the-art metaheuristic algorithms as it is centred on the unit operations of opening and closing a facility with no additional local search. We demonstrate that the proposed algorithm outperforms the current state-of-the-art metaheuristic algorithms for the OpM problem on standard datasets, with extremely low p-value $< 10^{-8}$.

The rest of the paper is organised as follows. Section 2 presents the formal problem definition and the basic notations used in the paper. Then the new proposed algorithm is described in Section 3. Section 4 explains the auxiliary data structures used in the implementation and analyses the time complexity of the facility opening and closure operators. Experimental results are reported in Section 5 including a thorough comparison with the existing state of the art on standard datasets. Finally, the paper is concluded with potential future work outlined in Section 6.

2. Notation and problem definition

Let $I = \{1, \ldots, n\}$ and $J = \{1, \ldots, m\}$ be sets of clients and facilities, respectively, and let d_{ij} be the distance between client $i \in I$ and facility $j \in J$. An instance of the OpM problem is then represented by (D, p), where $D = [d_{ij}]_{n\times m}$ is the distance matrix, and $p < m$ is a positive integer. The problem is to find the set P of p facilities that maximises the objective value

$$f\ (P) = \sum_{i\in I} \min_{j\in P} d_{ij}. \tag{1}$$

That is, we are maximising the sum of distances from each client to its nearest facility.

We assume that $p > 1$. Because the case $n \leq p < m$ is also trivial, we further assume $p < n$. Hence, $p \in \{2, \ldots, \min(n, m) - 1\}$.

Given a candidate solution P, we say the facilities in P are *open* and the remaining facilities in $J\backslash P$ are *closed*. We use $\Delta_{close}(P, j)$, or simply $\Delta_{close}(j)$ when no ambiguity arises, to denote the resulting increase in the objective value if the facility j becomes closed. That is, $\Delta_{close}\ (P, j) = f(P\backslash\{j\}) - f(P)$. Similarly, $\Delta_{open}(P, j)$ or $\Delta_{open}(j)$ is defined as $f(P) - f(P \cup \{j\})$. Note that these definitions have been made so that the values are nonnegative.

Algorithm 1. Main

Inputs: A distance matrix $D = [d_{ij}]_{n\times m}$
An Integer $p \in \{2, \ldots, \min(n, m) - 1\}$
Output: A set of p facilities
Parameters: $\gamma \in (0, 1)$ and $\tau > 0$

```
Begin
  Initialise P randomly with p facilities //and initialise related data structures
  bestP = P;  worstf = bestf = f = f(P);  radius = 1;  α = g(1)
  while termination_condition = false do
    IG1()
    IG2()
    // update radius and α:
    if bestf = worstf then
      gap = 0
    else
      gap = (bestf − f)/(bestf − worstf)
    end if
    radius = min{p − 1, radius + ⌊(p − 2) × gap⌋ + 1}
    α = g(radius)
  end while
  return bestP
End
```

3. Proposed algorithm

The new algorithm we propose in this paper (Algorithm 1) is a variant of the IG algorithm hybridised by a diversification mechanism similar to that used in VNS. It receives as input an instance (D, p) of the problem. It also has two parameters γ and τ whose roles are explained shortly. The algorithm returns as output the best solution to the corresponding OpM instance it finds.

3.1. Reconstruction size and algorithm parameters

As a trajectory (single point) algorithm (Blum and Roli, 2003), Algorithm 1 starts with an initial solution P and keeps changing it (hopefully improving it) during the search using an IG mechanism. Key to this is the variable *radius*, also known as the *reconstruction size*, which is the number of facilities to close during the deconstruction phase and to open during the construction phase. Another key variable in this operation is α, which holds the probability value by which each candidate is shortlisted when we choose the best facility for closure/opening. That is, if $\alpha = 1$, then we consider all potential facility changes, while if $\alpha = 0$, then we would not allow any facility to change.

We observed empirically that using a value for α that declines with the *radius*, as opposed to a constant α, speeds up the algorithm. Thus, in the proposed algorithm, we always determine α according to the following formula:

$$\alpha = g\ (radius) = \gamma 2^{-\tau(radius-1)} + (1-\gamma)\ rand(). \tag{2}$$

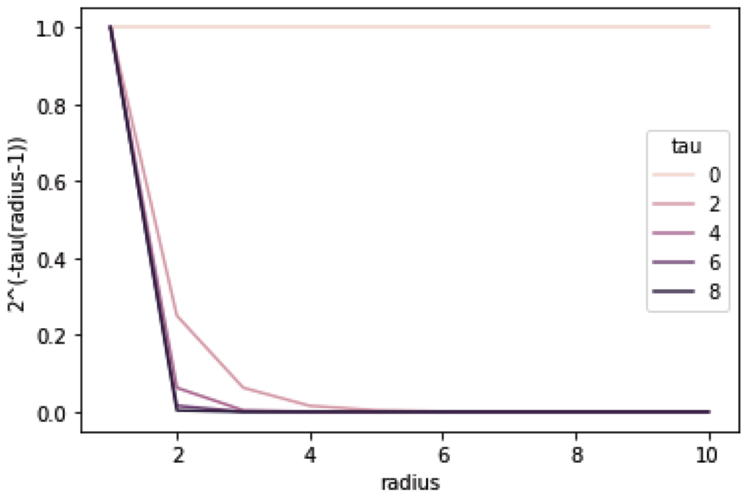

Fig. 1. Iluustration of the function $2^{-\tau(radius-1)}$ for $\tau \in \{0, 2, 4, 6, 8\}$ and $radius \in \{1, 2, \ldots, 10\}$. It is constant (1) for $\tau = 0$ and declines exponentially otherwise.

Here, γ and τ are parameters of the algorithm, and $rand()$ returns a nonnegative random value less than 1. In particular, the parameter τ controls how fast the function $2^{-\tau(radius-1)}$ declines as the $radius$ grows, while the parameter γ controls the balance between the value returned by this function and the randomisation introduced by $rand()$. Figure 1 depicts the function $2^{-\tau(radius-1)}$ for different values of τ.

3.2. Algorithm description

Algorithm 1 starts by initialising a random solution P (consisting of p facilities) in line 2. Then in line 3, it initialises the other variables: $bestP$, for storing the best solution found so far, is initialised to P; $bestf$ and $worstf$, for storing the best and worst objective values seen so far, are both initialised to $f = f(P)$; variable $radius$ is initialised to 1; and then α is set accordingly as described above.

The rest of Algorithm 1 consists of its main loop (lines 4–15), after whose termination the best solution found is returned (line 16). The loop termination condition could be a fixed number of iterations, a fixed number of consecutive iterations with no improvement, a time limit, or any combination of these, among other options.

Each iteration of the loop consists of two IG operations, **IG1** and **IG2** (lines 5–6), which we describe later. They may update the variables P, f, $best\ P$, $best f$, $worst f$, $radius$ and α. Also, there is control code (lines 8–14) to further update the variables $radius$ and α as follows. The temporary variable gap, calculated in lines 8–12, is used to indicate the quality of the current solution. It varies from 0 (when $f = bestf$) to 1 (when $f = worstf$) and is defined as 0 in the exceptional case when $f = worstf = bestf$. The variable $radius$ is then increased by the value $\lfloor (p-2) \times gap \rfloor + 1$ in line 13 unless its new value would exceed $p - 1$ in which case it becomes $p - 1$. The increment

Algorithm 2. IG1

Inputs: Uses P, f, $bestf$, $worstf$, $radius$ and α, as global variables
Outputs: Updates P, f, $best\ P$, $bestf$, $worstf$, $radius$ and α as global variables

```
1 Begin
2   improved = true
3   while improved do
4     improved = false
5     delta_f = 0
6     for d = 0 to radius do
7       delta_f = delta_f + Close_facility(P, α)
8     end for
9     for d = 0 to radius do
10      delta_f = delta_f − Open_facility(P, α)
11    end for
12    f = f + delta_f
13    if delta_f > 0 then
14      improved = true
15      radius = 1; α = g(1)
16      if f > bestf then
17        bestP = P; bestf = f
18      end if
19    end if
20    worstf = min{worstf, f}
21  end while
22 End
```

value $\lfloor (p-2) \times gap \rfloor + 1$ is at least 1 (when $gap = 0$) and at most $p - 1$ (when $gap = 1$). This diversification mechanism generalises that used in the standard VNS, where the radius is incremented by 1, by dynamically determining the increment based on the quality of the current solution. Finally, the variable α is updated based on the new value of $radius$ in line 14.

The IG operations **IG1** and **IG2** perform local searches: **IG1** searches by first closing and then opening facilities, while **IG2** searches by first opening and then closing facilities. These operations generalise, respectively, the local search operations RLS1 and RLS2, proposed by Herrán et al. (2020) and used by Gökalp (2020). They allow the reconstruction size (the variable *radius*) to be greater than 1 in a given search. **IG1** and **IG2** also contain randomisation on which facilities are considered for change, controlled by the probability value α. The IG algorithm of Gökalp (2020) also uses a reconstruction size (normally) greater than 1, but this is fixed during runtime, whereas in our proposed algorithm, it varies between 1 and $p - 1$ depending on the solution quality. Another difference is that the facility selection in the deconstruction phase in Gökalp (2020) is random not greedy, whereas the same (semi) greedy mechanism is used in both the deconstruction and the construction phases of the proposed algorithm.

Algorithm 2 presents the IG operation **IG1**. The algorithm for **IG2** is the same except that lines 7 and 10 are swapped (swapping whether we open or close facilities first). Therefore, we only describe **IG1** in detail. It consists of a **while**-loop (lines 3–21), which runs until no improved solution is found. At each iteration, it closes a number ($radius$) of facilities (the deconstruction phase in lines 6–8) and opens the same number of facilities (the construction phase in lines 9–11). This is achieved

Algorithm 3. Close_facility

Inputs: P and α
Outputs: Updates P and returns absolute change in the objective value
1 **Begin**
2 $df = -1$
3 **for** $j \in P$ **do**
4 **if** $random() < \alpha$ **then**
5 **if** $\Delta_{close}(j) > df$ **then**
6 $j^* = j$
7 $df = \Delta_{close}(j)$
8 **end if**
9 **end if**
10 **end for**
11 **if** $df = -1$ **then**
12 j^* = a random facility in P
13 $df = \Delta_{close}(j^*)$
14 **end if**
15 $P = P \setminus \{j^*\}$ **//and update related data**
16 **return** df
17 **End**

by invoking the **Close_facility** and **Open_facility** functions. These functions, which will shortly be described in more detail, close and open a single facility respectively, and return the absolute change in the objective value. After this, the objective value f is updated accordingly in line 12. If it has increased (as captured by the **if**-condition in line 13), then the flag *improved* is set to **true** (line 14) to allow for another iteration of the **while**-loop (lines 3–21). The variables *radius* and α are also set to 1 and $g(1)$, respectively (line 15), to increase the intensification. Further, if the obtained solution is even better than *bestP*, then *bestP* and *bestf* are updated (line 16–18). In the case where f becomes less than *worstf*, *worstf* is set to f (line 20).

The **Close_facility** (Algorithm 3) and **Open_facility** (Algorithm 4) operations are now described. The algorithm **Close_facility** receives as input a pair of P and α, closes a facility in P and returns the absolute amount of increase in the objective value. When $\alpha = 1$, the **for**-loop (lines 3–10) iterates through all the elements in P and finds a facility j^* whose closure would yield the maximum increase in the objective value. The amount of the increase $\Delta_{close}(j^*)$ is stored in the variable df, which will be returned in line 16. Note that the **for**-loop performs the selection process only, and the actual closure of j^* is performed in line 15. Notice that if $\alpha = 1$, then the condition of the **if**-statement in line 4 would always be **true**, which means all facilities would be allowed to 'compete' for the selection, and we would truly find the maximum increase in objective value. However, when $\alpha < 1$, then the selection process is not completely greedy because the **if**-statement in line 4 may filter out some facilities. In the (rare) case when all the facilities are filtered out (with probability $\alpha^{|P|}$), a facility will be selected uniformly at random (lines 11–14). The algorithm **Open_facility** (Algorithm 4) has a line-to-line correspondence with the algorithm **Close_facility** (Algorithm 3). It opens a facility among those outside P (and passed through the filter of line 4) that minimises the amount of decrease in the objective value.

Algorithm 4. Open_facility

```
Inputs:   J, P and α
Outputs:  Updates P and returns absolute change in the objective value
1 Begin
2   df = ∞
3   for j ∈ J \P do
4     if random() < α then
5       if Δ_open(j) < df then
6         j* = j
7         df = Δ_open(j)
8       end if
9     end if
10  end for
11  if df = ∞ then
12    j* = a random facility in J\P
13    df = Δ_open(j*)
14  end if
15  P = P ∪ {j*} //and update related data
16  return df
17 End
```

The randomness used in the **Close_facility** and **Open_facility** functions is a generalised variant of that used in the construction phase of GRASP proposed in Colmenar et al. (2016a). More specifically, Colmenar et al. (2016a) examined two different construction approaches, referred to as $C1$ and $C2$, which differ in the way the restricted candidate list (RCL) is generated and used. By $C1$, they meant the standard construction strategy used in the classic GRASP approach, where RCL is populated with the highest quality candidates from which one is selected randomly. In $C2$, the order of the greediness and randomness changes. There, RCL is populated randomly with a portion of the candidates and its best element is then selected (greedily). Colmenar et al. (2016a) showed that $C2$ would yield better results.

We also examined both approaches and observed the superiority of $C2$ in our early research. However, in contrast to Colmenar et al. (2016a) who use this mechanism for opening facilities only, we use it for facility closure as well. Furthermore, we use a variant of $C2$, in which the two steps of randomly populating RCL and finding its best element are performed simultaneously in a single loop (lines 3–10 of Algorithms 3 and 4).

4. Auxiliary data structures and facility opening and closure complexity

The proposed algorithm is built upon two unit operations of closing and opening a single facility, which are in turn based on two basic functions $\Delta_{close}(.)$ and $\Delta_{open}(.)$. Therefore, the efficiency of computing these functions is crucial to the overall running time of the algorithm. For this reason, we use auxiliary data structures to reduce their computational complexity as described in this section.

4.1. Auxiliary data structures

We use similar (but not identical) notations to those used in the literature of the p-center problem (Mladenović et al., 2003; Pullan, 2008; Mousavi, 2023). For each client i, $i = 1, \dots, n$, we keep the list N_i of all facilities sorted ascendingly by their distance from i, with ties broken arbitrarily. We use $N_i[k]$, $k = 1, \dots, m$, to refer to the kth facility in this list. We keep another list denoted as N_i^{-1} to record the location of a facility j, $j = 1, \dots, m$, in the list N_i. That is, $N_i^{-1}[j] = k$ if and only if $N_i[k] = j$. These two data structures are static, which means their data are fixed for a given problem instance. The other data structures described next are dynamic and their data change with P throughout a run of the algorithm. For a given P, we keep indicator variables x_j, $j = 1, \dots, m$, to indicate the membership of j in P, that is,

$$x_j = \begin{cases} 1 & j \in P \\ 0 & otherwise. \end{cases} \tag{3}$$

We also record for each client i a nearest facility F_i (to which i is assigned). If there is more than one nearest facility to i, we choose the one that appears first in the sorted list N_i. That is, $F_i = j$ if and only if $x_j = 1$ and $\forall j_1 : N_i^{-1}[j_1] < N_i^{-1}[j] \Rightarrow x_{j_1} = 0$. Finally, for each open facility j, we keep the set C_j of its assigned clients. That is, $C_j = \{i : F_i = j\}$. To the best of our knowledge, the data structures N_i^{-1} and C_j are novel in the literature of OpM and pM problems.

Proposition 1. *$\Delta_{close}(j) = \sum_{i\in C_j}(d_{iF_i^{(2)}} - d_{ij})$, where $F_i^{(2)}$ is the second-nearest facility to i.*

The proof follows by noting the facts that the closure of a facility j would not affect clients outside C_j and that it would replace F_i with $F_i^{(2)}$ for each client i in C_j.

Proposition 2. *$\Delta_{open}(j) = \sum_{i\in I:\, N_i^{-1}[j]<N_i^{-1}[F_i]}(d_{iF_i} - d_{ij})$.*

The proof follows by noting that opening a facility j will affect a client i only if j replaces its currently assigned facility F_i, which is the case if and only if j appears prior to F_i in N_i.

4.2. Complexity analysis

Let $p_t = |P|$. The following propositions provide the time complexity of computing $\Delta_{close}(.)$ and $\Delta_{open}(.)$ using the auxiliary data structures N_i, N_i^{-1}, x_j, F_i and C_j, $i = 1, \dots, n$, $j = 1, \dots, m$.

Proposition 3. *$\Delta_{close}(j)$ is computable in $O(nm/{p_t}^2)$ average time.*

Proof. *We use Proposition 1 to compute $\Delta_{close}(j)$. The average number of clients in C_j is n/p_t. To find $F_i^{(2)}$ for each client $i \in C_j$, we start from the location $N_i^{-1}[j]$ of facility j in N_i and proceed forward until we reach the location of another open facility. This takes $O(m/p_t)$ average time because p_t out of the m facilities in the list are open. Once $F_i^{(2)}$ is found, calculating $d_{iF_i^{(2)}} - d_{ij}$ takes $O(1)$ time.*

Proposition 4. *$\Delta_{open}(j)$ is computable in $O(n)$.*

Proof. *We use Proposition 2 to compute $\Delta_{open}(j)$. We go through each client $i \in I$ in $O(n)$ time, check the condition $N_i^{-1}[j] < N_i^{-1}[F_i]$ in $O(1)$ and, if the condition is met, calculate $d_{iF_i} - d_{ij}$ in $O(1)$.*

Algorithm 5. Populate_static_data

```
Inputs:   Uses I, J and D as global variables
Outputs:  Populates static auxiliary data structures as global variables
1  Begin
2  //N_i,  i = 1, ..., n
3  for each client i ∈ I do
4      N_i = list of all facilities j ∈ J sorted ascendingly by d_ij
5  end for
6  //N_i^{-1},  i = 1, ..., n
7  for each client i ∈ I do
8    for k = 1 to m do
9       N_i^{-1}[N_i[k]] = k
10   end for
11 end for
12 End
```

To benefit from the results of Propositions 3 and 4, we need to populate the static data structures N_i and N_i^{-1} before their first usage, initialise the dynamic data structures x_j, F_i, C_j, $i = 1, \ldots, n$, $j = 1, \ldots, m$ after initialising P (line 2 of Algorithm 1) and update them whenever P changes by closing or opening a facility (lines 15 of Algorithms 3 and 4). The pseudocodes of these four operations are presented in Algorithms 5–8, respectively.

The following time complexity analyses assume that the set C_j of clients assigned to facility j, $j = 1, \ldots, m$, is implemented so that the operations $C_j = \{\}$, $C_j = C_j \cup \{i\}$, and $C_j = C_j \setminus \{i\}$, $i = 1, \ldots, n$, are performed in $O(1)$. Otherwise, the complexity of these operations, which depend on the implementation, will need to be considered.

As can be seen in Algorithm 5, the time complexity of populating N_i^{-1} is dominated by that of N_i, $i = 1, \ldots, n$, (lines 3–5), which is $O(nm \log m)$ using merge sort or a similar algorithm.

The time complexity of initialising the dynamic data structures (Algorithm 6) is dominated by that of the last **for**-loop (lines 13–21), whose average running time is $O(nm/p)$ because it iterates n times and its inner **while**-loop (lines 16–18) takes $O(m/p)$ average time to reach an open facility in the list N_i. Note that p out of m facilities in N_i are open.

The average time complexity of updating the dynamic data structures after closing a facility j is the same as that of calculating $\Delta_{close}(j)$, which is $O(nm/{p_t}^2)$ (Proposition 3), because the average number of the iterations of the **for**-loop (lines 4–13 of Algorithm 7) is n/p_t and the **while**-loop (lines 7–9) takes $O(m/p_t)$ average time. Similarly, as can be seen in Algorithm 8, the average time complexity of updating the dynamic data structures after opening a facility is the same, $O(n)$, as that of computing $\Delta_{open}(.)$.

Given these results, it is not hard to see that the average running times of the **Close_facility** and **Open_facility** functions (Algorithms 3 and 4) are $O(nm/p_t)$ and $O((m-p)n)$, respectively. However, the time complexity analysis of **IG1** and **IG2** is non-trivial because the number of iterations of their **while**-loops (lines 3–20 of Algorithm 2), depends on the quality of the solutions found during the search. This makes hard the complexity analysis of the main algorithm. In addition, the number of iterations of the **while**-loop of the main algorithm (lines 4–15 of Algorithm 1) depends on

Algorithm 6. Initialise_dynamic_data

Inputs: P
Uses I, J and static auxiliary data structures as global variables
Outputs: Updates dynamic auxiliary data structures as global variables

Begin
  // $x_j,\ j = 1, \ldots, m$
  for each facility $j \in J$ **do**
    $x_j = 0$
  end for
  for each facility $j \in P$ **do**
    $x_j = 1$
  end for
  // F_i **and** C_j, $i = 1, \ldots, n$, $j = 1, \ldots, m$
  for each facility $j \in J$ **do**
    $C_j = \{\}$
  end for
  for each client $i \in I$ **do**
    //find F_i
    $k = 0$
    while $x_{N_i[k]} = 0$ **do**
      $k = k + 1$
    end while
    $F_i = N_i[k]$
    $C_{F_i} = C_{F_i} \cup \{i\}$
  end for
End

Algorithm 7. Update_dynamic_data_after_closure

Inputs: Facility j
Uses auxiliary data structures as global variables
Outputs: Updates dynamic auxiliary data structures as global variables

Begin
  // x_j
  $x_j = 0$
  for each client $i \in C_j$ **do**
    //set new F_i **to the current** $F_i^{(2)}$
    $k = 1 + N_i^{-1}[F_i]$
    while $x_{N_i[k]} = 0$ **do**
      $k = k + 1$
    end while
    $F_i = N_i[k]$
    //add i **to the new list**
    $C_{F_i} = C_{F_i} \cup \{i\}$
  end for
End

Algorithm 8. Update_dynamic_data_after_opening

Inputs: Facility j
Uses I and auxiliary data structures as global variables
Outputs: Updates dynamic auxiliary data structures as global variables

Begin
  //x_j
  $x_j = 1$
  $C_j = \{\}$
  for each client $i \in I$ **do**
    if $N_i^{-1}[j] < N_i^{-1}[F_i]$ **then**
      //remove i **from the old list**
      $C_{F_i} = C_{F_i} \setminus \{i\}$
      $F_i = j$
      //add i **to the new list**
      $C_j = C_j \cup \{i\}$
    end if
  end for
End

its termination condition, which could in turn be dependent on the solution quality (e.g., a fixed number of consecutive iterations without improvement).

5. Experimental results

To evaluate the performance of the proposed algorithm, it was implemented and compared with the current state-of-the-art metaheuristic techniques for the problem identified in Section 1.1. For simplicity, in the rest of this section, by TS and IG we mean the algorithms in Chang et al. (2021) and Gökalp (2020), respectively. We call our proposed algorithm IGV, standing for IG with variable reconstruction size.

We originally implemented IGV in Java. We obtained the original source code of TS and IG from the respective authors, which were written in Python and C++, respectively. In order to achieve a meaningful comparison free of the different programming language features, we then also implemented our algorithm IGV in Python and C++. This is especially important for a fair comparison of TS and IGV because a typical Python program can be significantly slower than its equivalent Java (or C++) version. The source code of IGV in these three languages is available at https://github.com/srm2022/opm.

Section 5.1 explains the datasets used in the benchmarking. In Section 5.2, the IGV parameters γ and τ are adjusted. The comparison of IGV with TS and IG is then presented in Sections 5.3 and 5.4, respectively. The parameter tuning experiments in Section 5.2 were performed using a laptop with Intel Core i5-6200 @ 2.3GHz CPU and 8 GB of RAM. The experiments in Sections 5.3 and 5.4 used the same desktop machine with an Intel Core i5-2400 CPU @ 3.10 GHz and 8 GB of RAM.

Table 1
Impact of parameter values on the running time of IGV

$\tau \setminus \gamma$	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1	104	157	171	172	179	131	123	70	79
2	112	78	90	77	71	61	42	39	123
3	103	66	89	65	46	39	56	44	160
4	106	96	64	53	53	44	45	51	167
5	97	76	72	79	54	56	43	66	156
6	105	76	77	66	58	**36**	37	94	192
7	86	77	83	66	50	42	38	50	168
8	114	88	69	62	57	42	41	61	201
9	116	92	89	56	78	45	59	63	237

5.1. Benchmark datasets

Two datasets were used. The first dataset is available in Colmenar et al. (2016b). It consists of 144 OpM instances used in the recent literature (Gökalp, 2020; Herrán et al., 2020; Mladenović et al., 2020). Each instance has the same number n of clients and facilities, which varies from 200 to 450. For each value of n, there are six instances, two per for each value of $p = \lfloor n/8 \rfloor$, $\lfloor n/4 \rfloor$ and $\lfloor n/2 \rfloor$. The benchmark consists of two lists of 72 instances, labelled with 'A' and 'B', with a one to one correspondence. Each instance in the latter was obtained by transposing the distance matrix of its corresponding instance in the former. The former list was obtained by modifying 24 instances of the pM problem (Reese, 2006; Belotti et al., 2007; Mladenović et al., 2007) available in the OR-Library (Beasley, 1990a, 1990b). In the rest of the paper, we will refer to this dataset as the *small dataset*.

The number of clients n in the small dataset is at most 450. To compare the algorithms on larger instances, we produced a second dataset using the pmed benchmarks from the OR-Library (Beasley, 1990a, 1990b). These instances were originally created for the pM problem, but they can readily be used for the OpM problem by assuming $I = J$. We used instances pmed21 to pmed40, which all have $n \geq 500$ clients, and we also changed the original values of p to $n/4$ and $n/3$ to make a more challenging benchmark of 40 instances. Each data file contains a weighted graph (in addition to the values of n and p), which needs to be converted to a complete graph. We used the Floyd–Warshall algorithm for this purpose. We will refer to this dataset as the *large dataset* in the rest of the paper.

5.2. Tuning of IGV parameters

To adjust the parameters γ and τ, we randomly selected 14 instances (approximately 10%) of the small dataset. We evaluated every pair (γ_i, τ_j) such that $\gamma_i \in \{0.1, 0.2, \ldots, 0.9\}$ and $\tau_j \in \{1, 2, \ldots, 9\}$, by running IGV 10 times on each of the 14 instances and observing the total running time (for the 140 runs) needed to achieve the best-known objective values in the literature (Gökalp, 2020; Chang et al., 2021). The running times, rounded to the nearest second, are reported in Table 1.

Table 1 suggests that the best pair is $(\gamma, \tau) = (0.6, 6)$, where the total running time is 36 seconds. The parameters are fixed to these values in the following experiments.

5.3. Comparison with the prior state-of-the-art TS

We ran the source code of TS with its original settings on both the small and the large datasets. As in Chang et al. (2021), it was run 10 times per instance.

5.3.1. Comparison with TS on small instances

Let f_i t_i and t'_i be, respectively, the best objective value, the hit time (i.e., the time taken to find f_i) and the total time of the ith run of TS ($i = 1,\ldots,10$) on a given instance. We adjusted the termination condition of IGV such that its ith run on that instance was terminated upon achieving a solution with a better or the same objective value as f_i.

The results are presented in Table 2. The first three columns describe the problem instance: First, we have the instance filename, which contains within it the number of facilities p. For brevity, the actual filename is shortened here. For example, instead of 'pmed17.txt.table.p100.A.txt', 'pmed17-p100.A' is used. Then the number of clients n (which equals the number of facilities) is presented, and finally the best-known objective value from the literature (Gökalp, 2020; Chang et al., 2021) is included. The next group of columns concerns the performance of TS: The first three columns report the average, the standard deviation and the best of the objective values obtained by TS over its 10 runs per instance; the subsequent two columns report the average and the standard deviation of the hit time t (in seconds); and the final two columns of that group report the average and the standard deviation of the total running time t' of the algorithm. The remaining group of columns reports the respective results for IGV excluding the average and the standard deviation of its total running time because it terminates upon hitting (or exceeding) the target objective values. The last row shows the averages.

The results of Table 2 allow us to conclude that IGV is significantly faster than TS in obtaining the same (or better) objective values. IGV has a smaller average hit time for 138 out of the 144 instances. The average hit time of IGV over all 144 instances is 6.89 seconds, compared to 32.72 seconds for TS. Using a one-tailed paired t-test, the null hypothesis that the average hit time of IGV (over 10 runs for each instance) is not less than that of TS is rejected with an extremely low p-value $< 6.7 \times 10^{-28}$.

The average hit times of the algorithms are visualised in Fig. 2a, where the instances are numbered as ordered in Table 2. Their side-by-side box plots are shown in Fig. 2b, which indicates lower average hit time percentiles for IGV.

5.3.2. Comparison with TS on large instances

Table 3 shows the results of the comparison on the large instances. As above, TS was run 10 times per instance using its original settings. However, this time IGV was run for the same amount of running time as spent by TS to observe whether or not it could find better objective values within that same time. The definitions of the columns in Table 3 are the same as those in Table 2 except that the third column reports p instead of *Best*. The *Best* values are not known for this dataset because, to the best of our knowledge, this is its first use for OpM. Please note that the average and standard deviation of the total running time, t', of IGV are not included as they are by design equivalent to those of TS. The last two columns report the average and the standard deviation of the times when it achieves (or exceeds) the best objective values obtained by TS.

Table 2
Comparison of TS and IGV (in Python) on small instances

Instance			TS							IGV (Python)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed17-p100.A	200	4054	4054	0	4054	12.14	7.16	61.7	0.98	4054.00	0.00	4054	0.69	0.17
pmed17-p25.A	200	7317	7317	0	7317	1.835	0.63	39.7	3.52	7317.00	0.00	7317	1.75	1.23
pmed17-p50.A	200	5411	5409	3.77	5411	44.36	17.3	60.7	0.93	5409.10	3.51	5411	6.19	5.50
pmed18-p100.A	200	4220	4219.4	0.92	4220	38.13	17.9	61.2	1.4	4219.70	0.64	4220	2.18	1.15
pmed18-p25.A	200	7432	7432	0	7432	1.507	0.53	39.1	5.34	7432.00	0.00	7432	0.96	0.63
pmed18-p50.A	200	5746	5746	0	5746	5.673	3.25	60.1	4.41	5746.00	0.00	5746	2.42	2.02
pmed19-p100.A	200	4033	4033	0	4033	38	14.2	60.9	0.9	4033.00	0.00	4033	2.48	1.83
pmed19-p25.A	200	7020	7020	0	7020	1.524	0.36	37.7	5.49	7020.00	0.00	7020	0.49	0.10
pmed19-p50.A	200	5387	5386.9	0.3	5387	27.98	21.5	60.7	0.63	5387.00	0.00	5387	10.35	9.36
pmed20-p100.A	200	4063	4062.7	0.46	4063	34.31	14.8	61.5	1.35	4062.80	0.40	4063	1.36	0.43
pmed20-p25.A	200	7648	7648	0	7648	1.522	0.75	39	6.72	7648.00	0.00	7648	0.52	0.12
pmed20-p50.A	200	5872	5872	0	5872	10.77	4.97	61.8	1.09	5872.00	0.00	5872	1.30	0.55
pmed21-p125.A	250	4155	4151.1	2.84	4153	27.28	12	61.1	0.98	4151.50	3.11	4155	4.38	3.91
pmed21-p31.A	250	7304	7304	0	7304	3.617	2.15	60.5	2.99	7304.00	0.00	7304	3.73	3.16
pmed21-p62.A	250	5784	5774.1	9.13	5784	48.8	16.5	61	0.88	5775.80	8.57	5784	8.16	7.03
pmed22-p125.A	250	4358	4342.2	7.24	4351	50.04	12.1	61.6	1.24	4342.90	7.01	4351	4.02	2.68
pmed22-p31.A	250	7900	7900	0	7900	4.14	1.65	60.9	2.9	7900.00	0.00	7900	1.76	1.08
pmed22-p62.A	250	5995	5995	0	5995	25.96	12.3	60.7	0.66	5995.00	0.00	5995	4.30	2.28
pmed23-p125.A	250	4114	4096.9	10.8	4114	54.88	6.14	61.6	1.54	4097.80	10.61	4114	11.40	14.35
pmed23-p31.A	250	7841	7841	0	7841	2.09	0.8	61.8	1.47	7841.00	0.00	7841	1.77	1.17
pmed23-p62.A	250	5785	5784.1	2.7	5785	23.01	14.3	61.6	0.96	5784.10	2.70	5785	3.62	2.44
pmed24-p125.A	250	4091	4088.2	4.75	4091	42.92	15.1	62.3	1.73	4089.00	4.20	4091	7.60	3.72
pmed24-p31.A	250	7425	7425	0	7425	2.587	1.25	62.1	1.1	7425.00	0.00	7425	2.14	1.39
pmed24-p62.A	250	5528	5525.1	5.72	5528	33.16	19.4	61.6	1	5526.20	4.49	5528	4.59	2.35
pmed25-p125.A	250	4155	4149.3	10.9	4155	39.17	18.9	61.9	1.63	4150.00	9.56	4155	13.88	15.65
pmed25-p31.A	250	7552	7552	0	7552	2.119	0.39	56.3	6.64	7552.00	0.00	7552	1.08	0.44
pmed25-p62.A	250	5767	5767	0	5767	28.78	16.8	61.3	1.1	5767.00	0.00	5767	3.45	1.39
pmed26-p150.A	300	4341	4319.9	5.72	4325	46.36	14.7	61.8	1.39	4321.90	6.64	4329	2.86	1.13
pmed26-p37.A	300	8112	8112	0	8112	2.367	0.28	62.6	1.92	8112.00	0.00	8112	1.29	0.26
pmed26-p75.A	300	5789	5787	4	5789	47.85	12.4	61.6	1.42	5787.10	3.81	5789	11.34	9.19
pmed27-p150.A	300	4062	4036.3	13.7	4054	50.68	7.41	61.5	1.13	4039.10	11.50	4054	4.69	2.90

Continued

Table 2
Continued

Instance			TS							IGV (Python)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed27-p37.A	300	7556	7556	0	7556	4.52	1.17	62.2	1.74	7556.00	0.00	7556	3.33	1.51
pmed27-p75.A	300	5668	5664.8	3.92	5668	39.04	18.4	61	0.88	5664.90	3.81	5668	20.34	19.89
pmed28-p150.A	300	4099	4064.4	18.1	4093	41.49	17.6	61.9	1.83	4066.70	17.73	4093	3.36	1.74
pmed28-p37.A	300	7366	7366	0	7366	3.449	0.47	62.2	1.57	7366.00	0.00	7366	2.45	1.22
pmed28-p75.A	300	5681	5678.7	4.61	5681	34.1	20.2	61.7	0.81	5678.80	4.42	5681	19.75	13.64
pmed29-p150.A	300	4141	4116.8	9.74	4135	41.78	9.63	62.2	2.03	4118.60	10.57	4140	2.33	0.62
pmed29-p37.A	300	7404	7404	0	7404	14.99	12	61.8	1.35	7404.00	0.00	7404	10.87	6.76
pmed29-p75.A	300	5880	5880	0	5880	33.04	18.2	62.3	1.48	5880.00	0.00	5880	5.23	2.19
pmed30-p150.A	300	4385	4378.7	5.37	4385	43.84	18	61	0.76	4380.00	5.55	4385	8.90	12.95
pmed30-p37.A	300	7704	7704	0	7704	4.661	2.29	63	1.88	7704.00	0.00	7704	1.96	0.83
pmed30-p75.A	300	6189	6183.3	4.78	6189	37.05	16.6	62.2	1.34	6184.50	4.52	6189	6.46	3.58
pmed31-p175.A	350	4136	4106.1	18.8	4129	55.62	6.3	62.6	1.62	4110.20	15.35	4130	5.13	3.34
pmed31-p43.A	350	7424	7424	0	7424	6.179	1.65	62.6	1.84	7424.00	0.00	7424	3.13	1.32
pmed31-p87.A	350	5905	5903.8	2.4	5905	33.17	13.6	63.4	2.25	5904.10	1.92	5905	16.49	12.01
pmed32-p175.A	350	4242	4198.5	19.6	4232	50.53	12.3	61.6	1.25	4200.70	20.72	4233	4.52	3.43
pmed32-p43.A	350	7794	7788.2	11.6	7794	16.98	16.6	62.5	1.36	7788.20	11.60	7794	17.28	19.75
pmed32-p87.A	350	5925	5905.2	7.56	5925	43.84	14.5	62.4	2.05	5906.70	7.20	5925	8.49	4.65
pmed33-p175.A	350	4105	4080.3	9.88	4100	57.33	10	63	2.44	4081.30	9.24	4101	4.49	2.16
pmed33-p43.A	350	7598	7598	0	7598	5.38	1.16	63.5	1.11	7598.00	0.00	7598	4.60	2.51
pmed33-p87.A	350	5793	5772.2	12.9	5790	50.38	15.2	61.3	0.72	5774.30	12.70	5790	7.01	4.38
pmed34-p175.A	350	4287	4237.7	22.7	4278	49.89	12.2	63.2	1.76	4239.90	23.12	4279	6.00	3.24
pmed34-p43.A	350	7725	7725	0	7725	7.013	3.64	63.1	2.16	7725.00	0.00	7725	6.22	3.53
pmed34-p87.A	350	5849	5835.8	6.26	5844	41.93	13.2	61.8	1.57	5838.40	5.31	5849	6.38	4.26
pmed35-p100.A	400	5845	5812.2	24.2	5845	53	10.4	61.8	1.22	5815.40	24.83	5845	7.66	4.63
pmed35-p200.A	400	4007	3956.7	12.1	3973	62.04	3.79	63.4	1.72	3958.50	10.79	3974	6.30	2.41
pmed35-p50.A	400	7155	7155	0	7155	22.08	8.79	62.6	1.12	7155.00	0.00	7155	11.83	7.02
pmed36-p100.A	400	6461	6456.6	4.96	6461	45.66	13.4	62.3	1.49	6457.70	5.14	6461	6.31	3.53
pmed36-p200.A	400	4319	4261.7	21.7	4296	58.66	9.87	65	1.92	4265.00	19.99	4296	5.44	3.94
pmed36-p50.A	400	8179	8178.4	1.8	8179	23.84	15.3	62.9	1.62	8178.40	1.80	8179	9.31	4.27
pmed37-p100.A	400	6203	6173.1	20.2	6196	52.91	11.1	62.1	1.14	6176.50	19.54	6203	17.16	18.94
pmed37-p200.A	400	4593	4531.4	13.9	4563	63.75	4.57	65.4	2.42	4534.90	12.14	4566	3.72	1.36

Continued

Table 2
Continued

Instance			TS							IGV (Python)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed37-p50.A	400	7830	7827.6	4.8	7830	34.5	13.5	62.5	1.76	7827.70	4.61	7830	8.62	3.91
pmed38-p112.A	450	5915	5898	4.63	5905	48.36	6.98	63.5	1.61	5898.60	5.30	5906	10.33	7.02
pmed38-p225.A	450	4428	4364.4	17.3	4388	106	7.18	110	5.38	4368.60	15.97	4388	5.85	1.94
pmed38-p56.A	450	7432	7432	0	7432	20.72	5.27	63.7	2.71	7432.00	0.00	7432	10.40	4.98
pmed39-p112.A	450	5935	5921.7	7.56	5935	54.43	7.78	62.9	1.55	5924.40	7.62	5935	12.41	6.14
pmed39-p225.A	450	4369	4317.2	17.6	4349	105.7	8.79	109	6.97	4322.60	15.00	4351	5.31	3.16
pmed39-p56.A	450	7712	7712	0	7712	22.63	6.22	62.8	2.66	7712.00	0.00	7712	6.36	3.09
pmed40-p112.A	450	6272	6249.8	12	6270	56.77	8.9	64.2	2.11	6251.40	11.89	6271	9.66	4.05
pmed40-p225.A	450	4572	4522.9	10.7	4536	102.3	8.3	105	5.39	4524.50	10.23	4539	7.71	4.06
pmed40-p56.A	450	8211	8207	8	8211	29.36	12.5	63	1.67	8207.60	6.93	8211	8.76	6.62
pmed17-p100.B	200	3992	3992	0	3992	8.594	5.23	61.4	1.32	3992.00	0.00	3992	0.80	0.48
pmed17-p25.B	200	6905	6905	0	6905	1.731	0.79	41	3.54	6905.00	0.00	6905	1.49	1.61
pmed17-p50.B	200	5563	5563	0	5563	15.02	9.14	61.6	0.79	5563.00	0.00	5563	2.29	2.67
pmed18-p100.B	200	4122	4119.5	2.94	4122	35.09	17.6	60.9	0.89	4120.00	2.45	4122	4.15	3.84
pmed18-p25.B	200	7662	7662	0	7662	1.651	0.57	40.9	3.64	7662.00	0.00	7662	0.61	0.24
pmed18-p50.B	200	5852	5852	0	5852	10.19	10	61.6	0.91	5852.00	0.00	5852	1.35	0.71
pmed19-p100.B	200	4016	4014.2	2.75	4016	27.36	16.7	61.2	0.98	4014.80	2.40	4016	2.73	1.80
pmed19-p25.B	200	6816	6816	0	6816	1.522	0.55	41.7	4.02	6816.00	0.00	6816	0.71	0.53
pmed19-p50.B	200	5423	5423	0	5423	9.806	8	61.6	1.04	5423.00	0.00	5423	1.74	0.71
pmed20-p100.B	200	4067	4066.7	0.46	4067	38.05	21.1	61.2	0.86	4066.80	0.40	4067	33.60	22.14
pmed20-p25.B	200	7349	7349	0	7349	1.36	0.42	36.6	5.77	7349.00	0.00	7349	0.80	0.38
pmed20-p50.B	200	5665	5665	0	5665	7.925	5.95	62.3	1.03	5665.00	0.00	5665	1.71	1.26
pmed21-p125.B	250	4033	4024.6	2.65	4029	42.25	18.1	61.3	1.49	4026.10	3.56	4032	5.68	4.50
pmed21-p31.B	250	7331	7331	0	7331	2.491	0.94	61.7	2.91	7331.00	0.00	7331	1.69	0.92
pmed21-p62.B	250	5870	5870	0	5870	11.91	6.12	62.3	1.34	5870.00	0.00	5870	2.93	1.26
pmed22-p125.B	250	4338	4322.2	10.8	4338	48.02	9.54	60.9	1.08	4323.60	9.96	4338	12.35	22.30
pmed22-p31.B	250	7695	7695	0	7695	2.654	1.18	55.9	8.61	7695.00	0.00	7695	1.02	0.36
pmed22-p62.B	250	6259	6259	0	6259	5.085	2.7	62.3	0.76	6259.00	0.00	6259	2.09	0.75
pmed23-p125.B	250	4095	4085	9.15	4095	51.44	6.94	61.5	1.34	4085.30	8.93	4095	14.12	18.98
pmed23-p31.B	250	7137	7137	0	7137	8.734	7.02	61.7	1.14	7137.00	0.00	7137	10.45	14.49
pmed23-p62.B	250	5724	5724	0	5724	29.53	15	60.8	0.47	5724.00	0.00	5724	4.30	2.88
pmed24-p125.B	250	4072	4061.6	8.45	4072	52.15	9.59	61	0.62	4062.60	7.55	4072	9.78	12.54
pmed24-p31.B	250	7190	7190	0	7190	2.434	0.44	54.6	8.48	7190.00	0.00	7190	1.83	0.71

Continued

Table 2
Continued

Instance			TS							IGV (Python)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed24-p62.B	250	5752	5742.6	11.6	5752	18.96	9.3	62.1	1.45	5743.50	10.60	5752	7.05	4.04
pmed25-p125.B	250	4233	4225.3	6.33	4233	38.9	12.1	62	2.16	4226.30	6.23	4233	4.76	5.82
pmed25-p31.B	250	7552	7552	0	7552	9.034	7.06	56.6	7.79	7552.00	0.00	7552	6.31	4.62
pmed25-p62.B	250	5692	5691	3	5692	25.97	15.6	61.3	0.54	5691.30	2.10	5692	9.46	10.03
pmed26-p150.B	300	4173	4144	13.3	4163	43.94	12.9	62.2	2.03	4148.20	9.10	4163	5.65	7.34
pmed26-p37.B	300	7643	7643	0	7643	3.998	1.86	63.4	1.7	7643.00	0.00	7643	4.81	4.21
pmed26-p75.B	300	5923	5923	0	5923	13.84	5.44	62.4	2.4	5923.00	0.00	5923	6.51	2.99
pmed27-p150.B	300	4144	4127	17	4144	44.83	18.5	62	1.37	4128.80	16.50	4144	7.71	6.13
pmed27-p37.B	300	7448	7448	0	7448	9.136	5.05	61.8	1.05	7448.00	0.00	7448	2.47	1.02
pmed27-p75.B	300	5844	5838	9.17	5844	37.92	18	61.8	1.12	5839.00	7.95	5844	13.62	15.70
pmed28-p150.B	300	4069	4036.2	13.3	4055	57.24	8	62.2	2.5	4038.00	12.90	4056	4.62	3.01
pmed28-p37.B	300	7388	7388	0	7388	2.917	1.11	62.6	1.63	7388.00	0.00	7388	2.12	0.99
pmed28-p75.B	300	5642	5625.9	11.3	5642	40.33	13.1	61.1	1.29	5628.60	10.87	5642	7.58	6.76
pmed29-p150.B	300	4157	4136.2	8.58	4157	51.25	9.93	61.4	1.39	4138.30	7.81	4157	5.60	5.25
pmed29-p37.B	300	7529	7529	0	7529	8.182	4.36	61.4	1.18	7529.00	0.00	7529	2.32	0.92
pmed29-p75.B	300	5709	5708.5	1.5	5709	37.75	14.4	61.7	1.58	5708.50	1.50	5709	13.28	11.06
pmed30-p150.B	300	4313	4245.1	11.4	4268	52.64	11.4	61.7	1.7	4248.00	11.07	4269	2.80	1.41
pmed30-p37.B	300	8048	8048	0	8048	3.562	1	62.7	2.9	8048.00	0.00	8048	1.61	0.52
pmed30-p75.B	300	6041	6039.5	4.5	6041	41.75	14	61.5	0.96	6039.70	3.90	6041	7.44	3.44
pmed31-p175.B	350	4138	4106.2	9.26	4119	43.33	13.2	62.1	0.7	4108.80	8.63	4125	3.96	1.94
pmed31-p43.B	350	7320	7320	0	7320	29.21	9.62	62.1	2.13	7320.00	0.00	7320	12.19	7.89
pmed31-p87.B	350	5621	5605.9	11.9	5621	39.13	15.1	61.8	1.02	5606.70	11.09	5621	16.54	18.56
pmed32-p175.B	350	4247	4198.3	14	4226	49.65	13.2	62.7	2.98	4200.90	13.40	4226	7.12	10.58
pmed32-p43.B	350	7899	7899	0	7899	6.212	2.01	62.4	1.13	7899.00	0.00	7899	9.95	7.69
pmed32-p87.B	350	5852	5800.7	19.5	5821	52.34	11.5	61.7	0.93	5808.60	18.75	5839	6.14	3.55
pmed33-p175.B	350	4156	4108.1	10.8	4127	48.86	14.1	61.8	1.95	4110.50	11.52	4128	3.36	1.51

Continued

Table 2
Continued

Instance			TS								IGV (Python)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}		f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed33-p43.B	350	7611	7611	0	7611	7.42	4.63	62.4	2.04		7611.00	0.00	7611	6.54	5.49
pmed33-p87.B	350	5840	5815.8	10.7	5827	41.46	15.7	61.7	0.88		5819.00	11.76	5830	9.99	7.32
pmed34-p175.B	350	4270	4245.2	14.9	4269	41.54	16.2	63.8	3.06		4248.80	13.13	4270	4.95	5.73
pmed34-p43.B	350	7514	7514	0	7514	5.875	3.46	63.5	1.29		7514.00	0.00	7514	3.53	2.65
pmed34-p87.B	350	5857	5849.4	5.66	5857	50.1	14.6	60.8	0.74		5850.70	5.98	5857	11.57	7.54
pmed35-p100.B	400	5639	5621.1	14.7	5639	40.99	16.4	63.4	2.38		5624.60	13.21	5639	21.17	23.41
pmed35-p200.B	400	4109	4048.6	18.1	4090	57.31	8.63	63.6	2.04		4053.70	16.35	4090	4.32	3.03
pmed35-p50.B	400	7570	7570	0	7570	10.97	4.68	62.7	1.91		7570.00	0.00	7570	6.11	7.19
pmed36-p100.B	400	6219	6186.4	24.5	6212	46.5	15.6	62	1.51		6190.30	21.26	6212	18.27	21.38
pmed36-p200.B	400	4321	4252.9	19.6	4284	53.56	8.63	62.5	1.67		4256.20	18.47	4285	3.94	1.19
pmed36-p50.B	400	8144	8144	0	8144	22.38	13.7	62	1.21		8144.00	0.00	8144	7.07	4.25
pmed37-p100.B	400	6212	6186.8	15	6210	49.9	11.9	62.1	0.91		6191.00	15.15	6211	16.62	11.44
pmed37-p200.B	400	4609	4530.3	25.2	4576	55.36	5.6	62.7	2.11		4539.90	25.78	4576	4.40	2.93
pmed37-p50.B	400	8379	8379	0	8379	20.67	5.52	62.2	1.4		8379.00	0.00	8379	6.96	3.57
pmed38-p112.B	450	5949	5918.8	27.9	5949	52.3	9.07	62.6	1.24		5921.70	26.85	5949	20.04	15.99
pmed38-p225.B	450	4446	4397.3	10.6	4413	95.89	7.59	101	5.9		4399.60	11.49	4419	6.62	4.56
pmed38-p56.B	450	7535	7535	0	7535	28.77	9.7	63.7	2.09		7535.00	0.00	7535	33.92	20.91
pmed39-p112.B	450	6198	6184.2	17	6198	51.11	11	62.6	1.99		6189.10	9.57	6198	13.15	12.32
pmed39-p225.B	450	4268	4218.5	11.1	4246	96.37	9.39	103	6.55		4220.30	11.71	4246	4.18	0.77
pmed39-p56.B	450	7625	7618.7	11	7625	38.11	12.1	62.3	1.46		7620.90	9.92	7625	12.59	7.57
pmed40-p112.B	450	6200	6165.4	17.1	6199	53.58	11	62.9	1.48		6166.60	17.45	6199	11.00	4.30
pmed40-p225.B	450	4532	4488.7	13.7	4509	99.74	10.7	104	7.51		4490.80	12.96	4510	5.00	1.56
pmed40-p56.B	450	8022	8021.2	0.6	8022	33.1	14.9	62.1	1.09		8021.40	0.66	8022	11.99	7.96
Average	**312.50**	**5884.26**	**5871.23**	**6.21**	**5879.74**	**32.72**	**9.59**	**62.46**	**2.07**		**5872.37**	**5.86**	**5880.31**	**6.89**	**5.32**

Table 3
Comparison of TS and IGV (in Python) on large instances

Instance			TS							IGV (Python)				
filename	n	p	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed21	500	125	7032.4	105.15	7188	62.04	5.59	64.95	2.4	7463.30	95.17	7686	12.18	7.44
pmed21	500	166	5659.8	77.9	5760	68.33	4.15	70.49	4.56	6112.50	87.33	6246	11.22	5.66
pmed22	500	125	7343.2	110.34	7568	57.84	6.15	63.18	1.75	7795.80	246.76	8250	18.39	12.26
pmed22	500	166	5931.7	50.19	6017	67.56	6.04	72.38	3.37	6395.70	80.12	6543	7.63	3.30
pmed23	500	125	7004.4	95.93	7205	55.83	4.79	62.92	1.76	7454.50	119.21	7660	11.13	5.77
pmed23	500	166	5804.4	54.21	5894	65.36	5.73	70.42	2.95	6116.30	88.87	6235	13.25	9.19
pmed24	500	125	7019.7	72	7105	60.42	5.84	64.22	2.11	7436.00	77.37	7537	9.03	2.58
pmed24	500	166	5673.9	51.1	5762	66.31	8.22	70.54	5.45	6182.70	63.90	6244	7.24	5.32
pmed25	500	125	7072.3	113.16	7236	58.4	7.77	64.45	2.24	7756.40	140.82	7933	13.23	9.62
pmed25	500	166	5637.6	56.21	5772	66.95	6.17	70.72	3.08	6219.80	74.07	6318	9.50	5.02
pmed26	600	150	7109	113.61	7298	102.45	6.91	106.7	5.28	7574.80	135.54	7772	20.49	12.38
pmed26	600	200	5661	68.6	5768	185.6	14.37	193.6	8.18	6109.20	119.19	6280	16.37	7.44
pmed27	600	150	6984.7	124.57	7216	103.71	7.4	108.1	4.29	7504.70	103.08	7664	24.34	25.61
pmed27	600	200	5657.5	64.39	5762	191.19	9.49	197	4.58	6105.60	54.85	6175	19.23	10.67
pmed28	600	150	6796.4	108	6951	104.63	4.59	106.3	3.22	7334.90	126.93	7536	16.99	7.79
pmed28	600	200	5460.1	64.69	5619	179.83	13.58	190.5	7.24	5986.60	39.79	6039	15.88	8.97
pmed29	600	150	6969.9	90.52	7194	97.35	8.41	100.5	6.43	7534.10	137.12	7776	15.30	8.28
pmed29	600	200	5701.1	51.54	5751	186.73	12.84	195.5	7.4	6192.80	58.87	6257	17.57	13.05
pmed30	600	150	7400.3	82.28	7553	103.26	7.18	106.3	6.13	7819.60	60.08	7932	17.41	18.11
pmed30	600	200	6062.2	52.77	6185	194.63	13.34	200.9	8.82	6502.90	41.24	6587	19.10	15.44
pmed31	700	175	6951.4	69.56	7056	257.83	14.55	262.4	9.76	7598.80	151.35	7826	26.76	18.69

Continued

Table 3
Continued

Instance			TS							IGV (Python)				
filename	n	p	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed31	700	233	5642.7	42.16	5714	628.69	27.31	642.4	21.3	6260.60	69.52	6372	15.63	6.66
pmed32	700	175	7276.4	58.37	7365	242.19	14.19	246.7	10.4	7912.00	71.99	8050	27.22	17.44
pmed32	700	233	5900	52.39	6038	596.66	48.5	612.9	29.7	6495.90	42.80	6553	20.23	9.99
pmed33	700	175	7217.4	89.05	7432	234.77	12.92	240.8	8.43	7755.50	83.04	7889	20.18	10.21
pmed33	700	233	5786.6	51.12	5872	595.28	41.17	614.7	22.3	6434.40	63.09	6568	26.81	16.20
pmed34	700	175	7100.5	88.78	7234	251.69	19.31	259.9	14.3	7677.50	174.07	7948	22.02	13.91
pmed34	700	233	5766.5	59.29	5879	635.94	43.28	653.8	24.9	6294.10	52.64	6382	20.06	12.49
pmed35	800	200	6923.6	65.48	7015	714.08	32.94	738	18.7	7557.70	75.98	7658	35.35	20.33
pmed35	800	266	5614.1	68.45	5728	1031.1	78.26	1051	49.7	6192.30	49.20	6247	32.23	15.96
pmed36	800	200	7542.3	120.99	7827	747.64	33.58	769.1	29.4	8186.90	47.04	8267	36.31	27.36
pmed36	800	266	6141.8	85.37	6323	1011.2	65.91	1040	37.3	6664.10	69.47	6770	32.90	21.11
pmed37	800	200	7645.2	74.42	7818	690.74	38.66	724.6	22.5	8294.40	63.56	8389	37.16	24.39
pmed37	800	266	6193.9	45.85	6260	1011.5	53.01	1039	25.2	6819.20	37.58	6892	20.81	7.92
pmed38	900	225	7210.7	104.62	7374	1175.4	54.86	1198	42.5	7833.60	59.26	7913	55.37	36.29
pmed38	900	300	5820.3	63.8	5907	1669.4	47.67	1682	42.3	6437.20	33.67	6501	35.63	11.46
pmed39	900	225	7236.1	81.57	7398	1168.6	52.8	1193	38.5	7887.20	63.09	8014	58.54	25.23
pmed39	900	300	5845.2	22	5866	1647.6	46.84	1664	42.4	6453.20	80.20	6564	48.20	16.51
pmed40	900	225	7831	85.51	7993	1184.5	51.09	1191	50.6	8451.70	54.83	8515	44.28	18.59
pmed40	900	300	6331	40.35	6403	1607.9	64.53	1642	40.7	6880.90	46.12	6950	42.32	16.80
Average	**670**	**195**	**6499**	**74.4**	**6632.7**	**479.5**	**25.0**	**491.1**	**16.8**	**7042.14**	**83.47**	**7173.45**	**23.84**	**13.54**

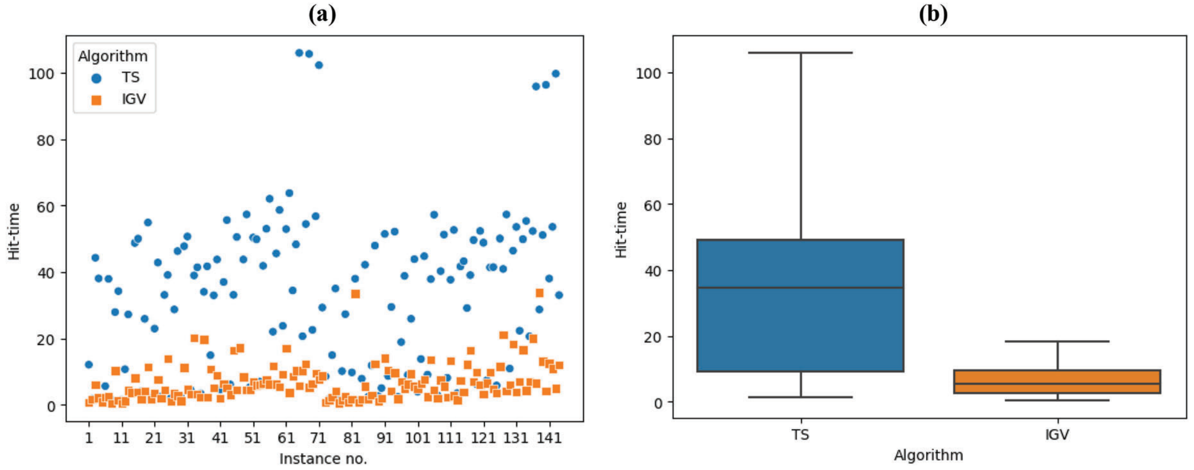

Fig. 2. (a) The average hit times of TS and IGV (Python) for small instances. (b) Their side-by-side box plots without outliers.

Table 3 shows that, for all 40 of these larger instances, IGV achieves improved average (and best) objective values. Using a one-tailed paired *t*-test, the null hypothesis that IGV does not improve the average objective values (f_{avg}) for these instances is rejected with an extremely low p-value $< 3.5 \times 10^{-33}$.

5.4. Comparison with the prior state-of-the-art IG

We now compare the implementation of our algorithms in C++ with the author's source code of IG. The comparison proceeds as in Section 5.3.

5.4.1. Comparison with IG on small instances

Table 4 presents the results for IG and IGV(C++) on the small problem instances, with the same columns as in Table 2.

Table 4 shows that, on average, IGV is significantly faster than IG in obtaining the same (or better) objective values. IGV has smaller average hit time for 141 out of 144 instances. Its average hit time over all 144 instances is 0.53 seconds, less than that of IG (26.29 seconds) by an order of magnitude. Using a one-tailed paired *t*-test, the null hypothesis that the average hit time of IGV (over 10 runs for each instance) is not less than that of IG is strongly rejected with a p-value $< 6.1 \times 10^{-8}$.

The average hit times of the algorithms are visualised in Fig. 3a. Their box plots are shown in Fig. 3b, which indicates significantly lower percentiles for IGV.

5.4.2. Comparison with IG on large instances

Table 5 shows the results for IG and IGV (C++) on the large instances, with the layout as in Table 3. Consistently with Section 5.3.2, IG was first run 10 times per instance using its original settings,

Table 4
Comparison of IG and IGV (in C++) on small instances

Instance			IG							IGV (C++)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed17-p100.A	200	4054	4054	0	4054	0.27	0.20	17.90	0.11	4054.00	0.00	4054	0.05	0.09
pmed17-p25.A	200	7317	7317	0	7317	0.15	0.12	3.16	0.02	7317.00	0.00	7317	0.04	0.04
pmed17-p50.A	200	5411	5411	0	5411	0.88	0.40	7.44	0.07	5411.00	0.00	5411	0.16	0.14
pmed18-p100.A	200	4220	4220	0	4220	0.46	0.26	17.88	0.30	4220.00	0.00	4220	0.14	0.10
pmed18-p25.A	200	7432	7432	0	7432	0.06	0.06	2.90	0.02	7432.00	0.00	7432	0.02	0.01
pmed18-p50.A	200	5746	5746	0	5746	0.16	0.11	7.37	0.07	5746.00	0.00	5746	0.05	0.04
pmed19-p100.A	200	4033	4033	0	4033	1.12	0.91	19.25	0.06	4033.00	0.00	4033	0.04	0.02
pmed19-p25.A	200	7020	7020	0	7020	0.01	0.01	2.99	0.03	7020.00	0.00	7020	0.01	0.01
pmed19-p50.A	200	5387	5386.1	0.3	5387	0.82	0.80	7.37	0.06	5386.60	0.49	5387	0.07	0.07
pmed20-p100.A	200	4063	4063	0	4063	0.89	0.34	17.96	0.17	4063.00	0.00	4063	0.05	0.03
pmed20-p25.A	200	7648	7648	0	7648	0.02	0.02	2.91	0.02	7648.00	0.00	7648	0.01	0.01
pmed20-p50.A	200	5872	5872	0	5872	0.46	0.29	7.00	0.05	5872.00	0.00	5872	0.06	0.05
pmed21-p125.A	250	4155	4154.8	0.6	4155	5.65	4.29	47.79	0.24	4155.00	0.00	4155	0.53	0.42
pmed21-p31.A	250	7304	7304	0	7304	0.55	0.31	6.66	0.04	7304.00	0.00	7304	0.05	0.03
pmed21-p62.A	250	5784	5783.5	0.67	5784	7.53	3.93	19.57	0.28	5783.70	0.46	5784	0.24	0.25
pmed22-p125.A	250	4358	4354.4	4.45	4358	10.62	10.84	39.98	0.54	4354.70	4.20	4358	0.51	0.62
pmed22-p31.A	250	7900	7900	0	7900	0.20	0.13	6.46	0.03	7900.00	0.00	7900	0.05	0.03
pmed22-p62.A	250	5995	5995	0	5995	1.02	1.03	17.10	0.07	5995.00	0.00	5995	0.09	0.06
pmed23-p125.A	250	4114	4114	0	4114	8.13	7.57	44.58	0.13	4114.00	0.00	4114	1.35	1.07
pmed23-p31.A	250	7841	7841	0	7841	0.04	0.05	6.84	0.02	7841.00	0.00	7841	0.04	0.04
pmed23-p62.A	250	5785	5785	0	5785	2.99	2.09	17.95	0.07	5785.00	0.00	5785	0.07	0.04
pmed24-p125.A	250	4091	4091	0	4091	9.32	6.24	49.98	0.09	4091.00	0.00	4091	0.06	0.03
pmed24-p31.A	250	7425	7425	0	7425	0.12	0.12	6.39	0.04	7425.00	0.00	7425	0.03	0.02
pmed24-p62.A	250	5528	5528	0	5528	1.15	1.35	16.83	0.08	5528.00	0.00	5528	0.12	0.06
pmed25-p125.A	250	4155	4155	0	4155	10.91	11.12	49.96	0.16	4155.00	0.00	4155	0.24	0.19
pmed25-p31.A	250	7552	7552	0	7552	0.06	0.05	6.73	0.03	7552.00	0.00	7552	0.02	0.01

Continued

Table 4
Continued

Instance			IG								IGV (C++)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed25-p62.A	250	5767	5767	0	5767	2.75	1.96	19.22	0.11	5767.00	0.00	5767	0.07	0.05
pmed26-p150.A	300	4341	4340.3	0.9	4341	40.14	23.30	94.62	0.73	4340.30	0.90	4341	1.98	1.79
pmed26-p37.A	300	8112	8112	0	8112	0.02	0.03	13.72	0.05	8112.00	0.00	8112	0.02	0.01
pmed26-p75.A	300	5789	5789	0	5789	4.21	2.34	38.03	0.14	5789.00	0.00	5789	0.19	0.13
pmed27-p150.A	300	4062	4062	0	4062	14.61	3.83	97.76	0.19	4062.00	0.00	4062	0.28	0.21
pmed27-p37.A	300	7556	7556	0	7556	0.65	0.37	13.96	0.04	7556.00	0.00	7556	0.06	0.03
pmed27-p75.A	300	5668	5668	0	5668	14.48	11.62	38.11	0.12	5668.00	0.00	5668	0.58	0.43
pmed28-p150.A	300	4099	4099	0	4099	8.99	5.05	82.21	0.31	4099.00	0.00	4099	0.52	0.22
pmed28-p37.A	300	7366	7366	0	7366	0.29	0.15	13.75	0.10	7366.00	0.00	7366	0.04	0.04
pmed28-p75.A	300	5681	5681	0	5681	6.19	4.40	38.58	0.21	5681.00	0.00	5681	0.45	0.48
pmed29-p150.A	300	4141	4139.1	1.45	4141	44.73	27.52	98.44	0.42	4139.90	1.51	4141	0.17	0.08
pmed29-p37.A	300	7404	7404	0	7404	1.05	0.81	12.91	0.04	7404.00	0.00	7404	0.21	0.18
pmed29-p75.A	300	5880	5880	0	5880	1.04	0.87	37.55	0.10	5880.00	0.00	5880	0.10	0.06
pmed30-p150.A	300	4385	4385	0	4385	3.72	2.63	88.78	0.72	4385.00	0.00	4385	0.42	0.49
pmed30-p37.A	300	7704	7704	0	7704	0.21	0.10	12.95	0.09	7704.00	0.00	7704	0.06	0.03
pmed30-p75.A	300	6189	6186.5	2.5	6189	7.95	6.92	38.62	0.26	6187.00	2.45	6189	0.20	0.17
pmed31-p175.A	350	4136	4135	0.45	4136	83.71	51.98	186.97	0.96	4135.30	0.64	4136	1.67	1.37
pmed31-p43.A	350	7424	7424	0	7424	0.58	0.31	23.26	0.13	7424.00	0.00	7424	0.07	0.04
pmed31-p87.A	350	5905	5905	0	5905	1.69	1.07	68.25	0.26	5905.00	0.00	5905	0.19	0.11
pmed32-p175.A	350	4242	4241.2	0.98	4242	39.75	43.93	160.43	0.65	4241.30	0.90	4242	0.69	0.87
pmed32-p43.A	350	7794	7794	0	7794	3.44	3.68	23.08	0.06	7794.00	0.00	7794	0.44	0.29
pmed32-p87.A	350	5925	5925	0	5925	7.58	8.34	66.53	0.17	5925.00	0.00	5925	0.45	0.33
pmed33-p175.A	350	4105	4102.5	1.75	4105	33.59	21.36	164.30	2.04	4103.00	1.48	4105	1.54	1.82
pmed33-p43.A	350	7598	7598	0	7598	0.40	0.39	23.27	0.09	7598.00	0.00	7598	0.12	0.15
pmed33-p87.A	350	5793	5793	0	5793	5.75	4.72	66.14	0.24	5793.00	0.00	5793	0.31	0.14
pmed34-p175.A	350	4287	4287	0	4287	17.74	19.27	168.99	0.22	4287.00	0.00	4287	0.85	0.58
pmed34-p43.A	350	7725	7725	0	7725	2.00	0.69	23.14	0.08	7725.00	0.00	7725	0.10	0.06

Continued

Table 4
Continued

Instance			IG							IGV (C++)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed34-p87.A	350	5849	5846.5	2.5	5849	11.46	10.04	67.23	0.52	5847.30	2.24	5849	1.57	1.29
pmed35-p100.A	400	5845	5845	0	5845	24.75	12.17	122.41	0.33	5845.00	0.00	5845	0.34	0.20
pmed35-p200.A	400	4007	4005.5	1.5	4007	105.77	70.85	314.22	1.73	4005.80	1.54	4007	1.41	1.14
pmed35-p50.A	400	7155	7155	0	7155	1.06	0.40	40.17	0.19	7155.00	0.00	7155	0.16	0.10
pmed36-p100.A	400	6461	6461	0	6461	8.99	3.22	115.52	0.32	6461.00	0.00	6461	0.12	0.07
pmed36-p200.A	400	4319	4316.3	4.22	4319	124.90	68.36	292.75	1.80	4316.60	4.08	4319	6.28	7.36
pmed36-p50.A	400	8179	8179	0	8179	1.40	1.02	39.13	0.13	8179.00	0.00	8179	0.10	0.07
pmed37-p100.A	400	6203	6203	0	6203	42.76	34.37	114.78	0.29	6203.00	0.00	6203	0.92	0.58
pmed37-p200.A	400	4593	4591.4	2.33	4593	188.30	86.45	302.35	2.60	4591.40	2.33	4593	1.19	1.32
pmed37-p50.A	400	7830	7830	0	7830	3.78	2.31	38.64	0.20	7830.00	0.00	7830	0.23	0.11
pmed38-p112.A	450	5915	5914.2	1.33	5915	77.54	51.80	197.86	2.25	5914.20	1.33	5915	2.20	2.03
pmed38-p225.A	450	4428	4426.7	1.1	4428	228.99	170.09	513.18	2.82	4427.00	1.00	4428	0.90	0.72
pmed38-p56.A	450	7432	7432	0	7432	0.80	0.62	63.16	0.30	7432.00	0.00	7432	0.18	0.08
pmed39-p112.A	450	5935	5935	0	5935	14.59	8.56	194.75	0.53	5935.00	0.00	5935	0.29	0.19
pmed39-p225.A	450	4369	4368.6	0.49	4369	202.38	111.73	491.95	1.10	4368.70	0.46	4369	5.97	4.45
pmed39-p56.A	450	7712	7712	0	7712	1.56	1.12	65.71	0.31	7712.00	0.00	7712	0.13	0.08
pmed40-p112.A	450	6272	6271.9	0.3	6272	76.38	53.88	193.10	1.07	6271.90	0.30	6272	1.27	0.71
pmed40-p225.A	450	4572	4570.3	1.35	4572	176.95	108.69	480.60	1.91	4570.50	1.20	4572	1.33	1.01
pmed40-p56.A	450	8211	8211	0	8211	1.91	0.61	67.30	0.18	8211.00	0.00	8211	0.14	0.06
pmed17-p100.B	200	3992	3992	0	3992	0.24	0.27	20.43	0.07	3992.00	0.00	3992	0.02	0.03
pmed17-p25.B	200	6905	6905	0	6905	0.06	0.05	2.83	0.01	6905.00	0.00	6905	0.02	0.01
pmed17-p50.B	200	5563	5563	0	5563	0.57	0.37	7.98	0.04	5563.00	0.00	5563	0.04	0.04
pmed18-p100.B	200	4122	4121.7	0.9	4122	2.41	1.86	16.28	0.19	4121.70	0.90	4122	0.06	0.04
pmed18-p25.B	200	7662	7662	0	7662	0.10	0.07	2.83	0.02	7662.00	0.00	7662	0.01	0.01
pmed18-p50.B	200	5852	5852	0	5852	0.18	0.16	6.94	0.06	5852.00	0.00	5852	0.03	0.02
pmed19-p100.B	200	4016	4016	0	4016	0.33	0.30	17.39	0.05	4016.00	0.00	4016	0.12	0.12
pmed19-p25.B	200	6816	6816	0	6816	0.02	0.01	2.69	0.02	6816.00	0.00	6816	0.01	0.00

Continued

Table 4
Continued

Instance			IG							IGV (C++)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed19-p50.B	200	5423	5423	0	5423	0.27	0.18	7.38	0.03	5423.00	0.00	5423	0.03	0.02
pmed20-p100.B	200	4067	4067	0	4067	2.30	2.15	17.88	0.06	4067.00	0.00	4067	0.48	0.43
pmed20-p25.B	200	7349	7349	0	7349	0.03	0.02	2.83	0.02	7349.00	0.00	7349	0.02	0.01
pmed20-p50.B	200	5665	5665	0	5665	0.12	0.11	7.33	0.06	5665.00	0.00	5665	0.04	0.03
pmed21-p125.B	250	4033	4032	3	4033	18.12	10.03	43.36	1.42	4032.30	2.10	4033	0.35	0.30
pmed21-p31.B	250	7331	7331	0	7331	0.12	0.09	6.63	0.03	7331.00	0.00	7331	0.03	0.03
pmed21-p62.B	250	5870	5870	0	5870	0.56	0.50	17.96	0.10	5870.00	0.00	5870	0.07	0.06
pmed22-p125.B	250	4338	4337.2	0.98	4338	17.78	12.61	43.68	0.18	4337.30	0.90	4338	0.48	0.64
pmed22-p31.B	250	7695	7695	0	7695	0.09	0.08	6.49	0.05	7695.00	0.00	7695	0.03	0.02
pmed22-p62.B	250	6259	6259	0	6259	0.26	0.20	18.66	0.08	6259.00	0.00	6259	0.03	0.03
pmed23-p125.B	250	4095	4095	0	4095	3.18	2.85	42.35	0.25	4095.00	0.00	4095	0.33	0.16
pmed23-p31.B	250	7137	7137	0	7137	0.35	0.48	6.11	0.06	7137.00	0.00	7137	0.22	0.16
pmed23-p62.B	250	5724	5724	0	5724	1.41	0.89	16.57	0.10	5724.00	0.00	5724	0.08	0.06
pmed24-p125.B	250	4072	4072	0	4072	5.15	4.19	47.13	0.13	4072.00	0.00	4072	0.16	0.09
pmed24-p31.B	250	7190	7190	0	7190	0.23	0.21	5.95	0.04	7190.00	0.00	7190	0.04	0.03
pmed24-p62.B	250	5752	5752	0	5752	6.79	4.51	17.35	0.18	5752.00	0.00	5752	0.20	0.15
pmed25-p125.B	250	4233	4230.2	2.86	4233	11.88	12.01	43.17	0.29	4230.70	2.45	4233	0.08	0.04
pmed25-p31.B	250	7552	7552	0	7552	0.66	0.55	6.85	0.04	7552.00	0.00	7552	0.20	0.14
pmed25-p62.B	250	5692	5692	0	5692	6.42	5.43	19.09	0.13	5692.00	0.00	5692	0.21	0.22
pmed26-p150.B	300	4173	4173	0	4173	19.21	15.63	99.55	0.22	4173.00	0.00	4173	0.25	0.16
pmed26-p37.B	300	7643	7643	0	7643	0.43	0.42	13.25	0.06	7643.00	0.00	7643	0.08	0.05
pmed26-p75.B	300	5923	5923	0	5923	2.96	2.67	38.50	0.29	5923.00	0.00	5923	0.06	0.03
pmed27-p150.B	300	4144	4144	0	4144	17.44	14.86	97.37	0.18	4144.00	0.00	4144	0.42	0.28
pmed27-p37.B	300	7448	7448	0	7448	0.34	0.20	13.26	0.05	7448.00	0.00	7448	0.05	0.03
pmed27-p75.B	300	5844	5844	0	5844	3.30	2.20	40.19	0.18	5844.00	0.00	5844	0.30	0.24
pmed28-p150.B	300	4069	4069	0	4069	39.67	20.23	92.64	0.19	4069.00	0.00	4069	0.30	0.21
pmed28-p37.B	300	7388	7388	0	7388	0.07	0.07	13.25	0.06	7388.00	0.00	7388	0.04	0.02

Continued

Table 4
Continued

Instance			IG							IGV (C++)				
Filename	n	*Best*	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed28-p75.B	300	5642	5640.1	3.81	5642	12.28	8.66	37.38	0.14	5640.10	3.81	5642	0.49	0.34
pmed29-p150.B	300	4157	4157	0	4157	5.73	2.38	103.64	1.94	4157.00	0.00	4157	0.22	0.15
pmed29-p37.B	300	7529	7529	0	7529	0.17	0.11	13.14	0.07	7529.00	0.00	7529	0.07	0.03
pmed29-p75.B	300	5709	5709	0	5709	10.37	8.41	36.98	0.18	5709.00	0.00	5709	0.26	0.18
pmed30-p150.B	300	4313	4313	0	4313	22.93	15.16	96.96	0.47	4313.00	0.00	4313	0.52	0.32
pmed30-p37.B	300	8048	8048	0	8048	0.38	0.39	13.31	0.17	8048.00	0.00	8048	0.03	0.02
pmed30-p75.B	300	6041	6041	0	6041	4.52	3.12	37.52	0.19	6041.00	0.00	6041	0.08	0.05
pmed31-p175.B	350	4138	4138	0	4138	28.67	17.28	180.12	0.58	4138.00	0.00	4138	1.25	0.59
pmed31-p43.B	350	7320	7320	0	7320	3.17	2.31	23.88	0.16	7320.00	0.00	7320	0.18	0.12
pmed31-p87.B	350	5621	5617.4	3.26	5621	30.30	15.45	70.01	0.23	5617.70	3.07	5621	0.54	0.53
pmed32-p175.B	350	4247	4243.7	0.9	4244	68.85	45.82	165.29	1.57	4244.70	1.79	4247	0.94	0.81
pmed32-p43.B	350	7899	7899	0	7899	3.26	5.88	23.60	0.17	7899.00	0.00	7899	0.22	0.22
pmed32-p87.B	350	5852	5845.2	3.52	5852	27.25	19.29	67.02	0.25	5847.70	3.66	5852	0.87	0.55
pmed33-p175.B	350	4156	4154.9	1.7	4156	65.87	41.12	173.18	1.21	4155.10	1.37	4156	0.63	0.28
pmed33-p43.B	350	7611	7611	0	7611	1.57	0.86	22.63	0.12	7611.00	0.00	7611	0.18	0.10
pmed33-p87.B	350	5840	5839.2	1.6	5840	27.54	20.27	67.73	0.31	5839.20	1.60	5840	0.51	0.43
pmed34-p175.B	350	4270	4270	0	4270	15.92	9.95	182.11	1.07	4270.00	0.00	4270	0.38	0.36
pmed34-p43.B	350	7514	7514	0	7514	0.39	0.28	24.21	0.10	7514.00	0.00	7514	0.06	0.04
pmed34-p87.B	350	5857	5855.7	1.35	5857	24.51	18.32	69.95	0.28	5856.20	1.25	5857	0.51	0.34
pmed35-p100.B	400	5639	5639	0	5639	26.37	23.00	115.73	0.71	5639.00	0.00	5639	0.79	0.45
pmed35-p200.B	400	4109	4108.3	1.19	4109	126.49	96.34	300.89	2.66	4108.30	1.19	4109	1.41	0.92
pmed35-p50.B	400	7570	7570	0	7570	0.89	0.37	42.20	0.27	7570.00	0.00	7570	0.15	0.10
pmed36-p100.B	400	6219	6214.3	3.00	6219	28.06	16.35	114.68	0.48	6214.70	3.03	6219	1.73	3.07
pmed36-p200.B	400	4321	4318.4	1.5	4321	113.80	67.75	264.96	5.36	4319.00	1.61	4321	1.33	1.15
pmed36-p50.B	400	8144	8144	0	8144	0.77	0.51	39.69	0.25	8144.00	0.00	8144	0.22	0.12
pmed37-p100.B	400	6212	6209.2	1.99	6212	66.97	30.43	113.50	0.95	6209.40	2.06	6212	1.36	1.07
pmed37-p200.B	400	4609	4608.6	0.8	4609	113.79	94.86	317.46	1.23	4608.60	0.80	4609	0.35	0.36
pmed37-p50.B	400	8379	8379	0	8379	0.90	0.51	39.19	0.18	8379.00	0.00	8379	0.43	0.99
pmed38-p112.B	450	5949	5949	0	5949	65.23	37.18	199.65	0.56	5949.00	0.00	5949	0.50	0.40
pmed38-p225.B	450	4446	4443.9	2.17	4446	311.32	184.21	566.25	4.40	4444.30	1.79	4446	0.95	0.57
pmed38-p56.B	450	7535	7535	0	7535	7.50	10.63	67.94	0.39	7535.00	0.00	7535	0.48	0.51
pmed39-p112.B	450	6198	6198	0	6198	55.33	55.61	198.66	0.64	6198.00	0.00	6198	0.39	0.33
pmed39-p225.B	450	4268	4264.1	2.34	4266	225.23	177.94	526.57	6.26	4264.30	2.24	4266	2.27	2.59
pmed39-p56.B	450	7625	7625	0	7625	3.95	2.51	65.29	0.25	7625.00	0.00	7625	0.28	0.13
pmed40-p112.B	450	6200	6199.6	0.92	6200	83.50	64.81	190.87	0.49	6199.60	0.92	6200	2.16	1.46
pmed40-p225.B	450	4532	4530.3	2.1	4532	329.04	118.57	497.58	1.72	4530.30	2.10	4532	8.00	6.88
pmed40-p56.B	450	8022	8022	0	8022	5.05	7.39	64.37	0.54	8022.00	0.00	8022	0.37	0.19
Average	**312.50**	**5884.26**	**5883.77**	**0.48**	**5884.22**	**26.29**	**16.92**	**82.47**	**0.50**	**5883.86**	**0.46**	**5884.24**	**0.53**	**0.46**

Table 5
Comparison of IG and IGV (in C++) on large instances

Instance			IG							IGV (C++)				
filename	n	p	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed21	500	125	7589.7	119.40	7669	161.04	72.29	293.93	3.15	7711.00	0.00	7711	3.36	3.30
pmed21	500	166	6195.6	73.36	6252	319.46	113.83	437.96	5.04	6281.30	7.46	6287	14.27	16.10
pmed22	500	125	8150	126.20	8245	177.26	70.59	287.59	9.01	8274.00	0.00	8274	2.76	1.70
pmed22	500	166	6468.4	76.35	6608	275.41	111.79	421.37	14.30	6628.00	0.00	6628	2.25	1.02
pmed23	500	125	7563.7	72.04	7642	169.80	30.23	277.84	5.21	7707.00	0.00	7707	2.39	1.69
pmed23	500	166	6257.5	35.13	6300	230.20	99.06	417.37	11.00	6312.00	8.97	6319	48.57	59.61
pmed24	500	125	7547.8	31.50	7588	173.77	64.24	280.09	5.85	7631.60	8.40	7642	9.80	12.80
pmed24	500	166	6219.2	57.54	6276	224.69	102.38	432.43	12.20	6281.90	4.06	6289	26.68	36.05
pmed25	500	125	7750.4	83.66	7857	197.76	76.57	303.51	4.05	7934.00	0.00	7934	1.85	1.58
pmed25	500	166	6179.7	65.42	6315	285.64	97.62	440.09	13.80	6395.30	3.74	6399	1.25	0.85
pmed26	600	150	7759.9	54.29	7808	482.34	180.16	752.07	13.00	7834.90	2.98	7836	62.46	76.85
pmed26	600	200	6192.4	61.33	6298	593.02	263.02	1108.70	21.70	6342.40	2.29	6344	16.09	34.84
pmed27	600	150	7557.9	58.14	7633	546.01	96.85	760.73	19.10	7709.00	0.00	7709	2.47	1.91
pmed27	600	200	6110.7	39.14	6175	797.63	281.13	1127.40	34.80	6232.00	3.92	6240	3.94	3.52
pmed28	600	150	7458.6	54.62	7535	533.20	165.86	720.81	16.10	7579.80	3.60	7581	6.34	5.15
pmed28	600	200	6057.8	35.97	6098	675.22	295.53	1100.80	33.70	6156.90	0.30	6157	7.34	4.70
pmed29	600	150	7801.6	81.48	7883	586.07	132.16	726.59	7.46	7905.00	0.00	7905	6.83	4.88
pmed29	600	200	6247.7	38.76	6329	833.94	270.19	1114.70	16.80	6366.20	1.60	6367	14.45	20.96
pmed30	600	150	7968.1	72.37	8068	505.81	188.56	752.89	9.64	8098.20	0.40	8099	8.41	6.70
pmed30	600	200	6553.2	36.31	6600	701.01	278.64	1146.10	39.60	6644.30	10.93	6658	10.63	15.94
pmed31	700	175	7621.6	117.80	7801	1029.04	504.02	1649.10	25.40	7862.00	0.00	7862	8.77	6.76
pmed31	700	233	6215.7	95.56	6355	1871.97	554.79	2624.00	31.60	6388.90	0.30	6389	13.69	15.66
pmed32	700	175	7914.7	52.34	7988	985.29	457.87	1609.60	34.40	8075.50	2.54	8079	8.11	7.03
pmed32	700	233	6496.6	44.70	6578	1386.35	494.75	2480.10	56.20	6606.70	8.04	6616	27.05	29.06

Continued

Table 5
Continued

Instance			IG							IGV (C++)				
filename	n	p	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}	t'_{avg}	t'_{std}	f_{avg}	f_{std}	f_{best}	t_{avg}	t_{std}
pmed33	700	175	7820.4	33.66	7888	1023.92	331.69	1629.50	9.97	7967.60	0.80	7968	4.66	1.77
pmed33	700	233	6416.5	58.38	6480	1478.08	603.38	2536.60	78.20	6579.10	1.30	6582	6.99	4.08
pmed34	700	175	7707.4	74.53	7863	1162.92	508.27	1676.30	54.60	7950.00	0.00	7950	8.86	15.32
pmed34	700	233	6302.7	32.69	6377	1511.73	614.08	2446.60	54.40	6411.90	5.50	6416	11.03	5.19
pmed35	800	200	7519	42.37	7580	2946.42	539.47	4086.80	92.30	7675.80	1.83	7677	10.81	6.49
pmed35	800	266	6138.1	34.72	6203	3399.07	825.30	4626.90	103.00	6253.70	2.69	6257	14.36	17.25
pmed36	800	200	8196.6	79.35	8283	1778.23	802.97	3229.30	50.90	8325.90	1.30	8327	21.94	14.32
pmed36	800	266	6683.5	51.42	6745	2804.68	869.03	4758.40	62.30	6807.40	5.66	6817	24.63	12.45
pmed37	800	200	8330.1	47.83	8420	2210.35	651.29	2989.20	66.90	8440.60	2.73	8443	92.53	222.80
pmed37	800	266	6776.8	55.08	6834	3075.17	1466.10	4615.70	175.00	6927.70	1.19	6929	28.17	20.73
pmed38	900	225	7794.2	66.24	7878	3420.46	1897.50	6239.10	115.00	7963.00	0.00	7963	16.04	9.46
pmed38	900	300	6325.6	65.06	6417	5284.34	2930.10	9478.50	119.00	6513.80	3.57	6520	15.23	12.08
pmed39	900	225	7902.1	105.00	8008	3410.52	1804.20	6255.10	78.50	8054.00	0.00	8054	24.41	18.91
pmed39	900	300	6378.3	67.62	6452	6903.65	1775.60	9327.20	226.00	6570.70	0.46	6571	17.61	9.68
pmed40	900	225	8452.3	67.29	8516	3814.37	1692.90	6009.70	61.10	8581.80	2.40	8584	27.86	15.09
pmed40	900	300	6858.7	31.92	6912	6452.64	1999.40	9336.10	224.00	6994.00	2.37	6998	16.36	12.76
Average	**670**	**195**	**7087**	**62.4**	**7168.9**	**1610.5**	**607.8**	**2512.7**	**50.4**	**7224.37**	**2.53**	**7227.20**	**16.28**	**19.18**

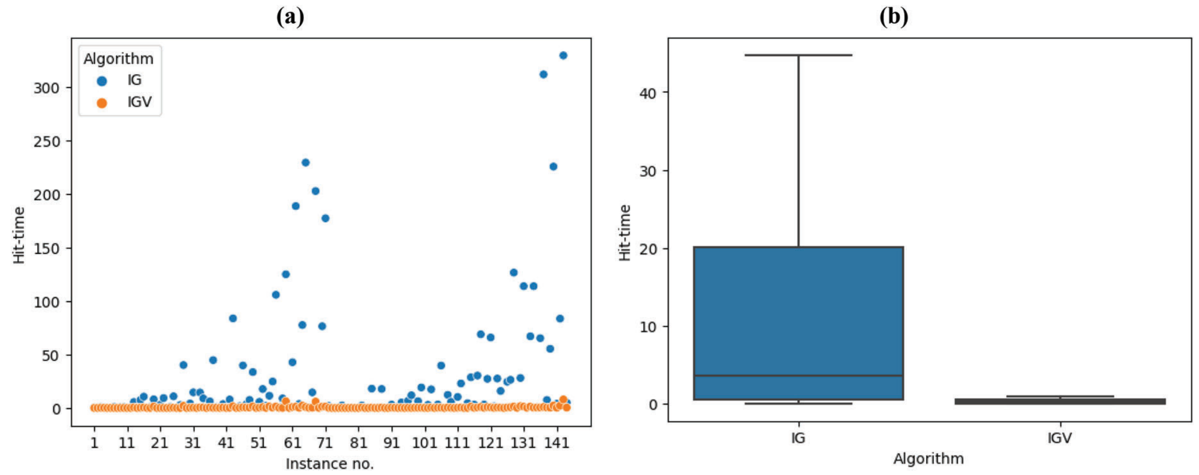

Fig. 3. (a) The average hit times of IG and IGV (C++) for small instances. (b) Their side-by-side box plots without outliers.

and then IGV was run for the same amount of running time as spent by IG on each run of each instance, allowing a fair comparison of the obtained objective values.

Table 5 reports that, for all 40 instances, IGV obtains improved average (and best) objective values, compared to those of IG. Using a one-tailed paired t-test, the null hypothesis that IGV does not improve the average objective values (f_{avg}) is rejected with a p-value $< 1.3 \times 10^{-22}$.

6. Conclusions and future work

6.1. Conclusions

This paper proposed a new algorithm for the OpM problem. It hybridises techniques from the IG and VNS metaheuristics, which are among the effective metaheuristics for optimisation problems (Demir, 2022; Rocha et al., 2022). It generalises previous ideas in the literature such as the reduced local search (Herrán et al., 2020) and the replacement of the facility swap operation with two consecutive operations of closing and opening a facility (Lin and Guan, 2018; Herrán et al., 2020). The main structure is an improved hybrid of those used in the IG algorithm by Gökalp (2020) and the standard VNS as detailed in Section 3. The overall algorithm is still simpler than most existing metaheuristic algorithms for the problem, being centred on two unit operations of closing and opening a facility with no additional local search.

The proposed algorithm significantly outperformed the current state-of-the-art metaheuristic algorithms on existing benchmark instances, achieving better or the same objective values in far less time. We also introduced a new benchmark set of larger instances upon which the new algorithm was found to achieve better objective values than the current stateoftheart when allowed the same time. We thus conclude the proposed algorithm to be the new state-of-the-art metaheuristic algorithm for the OpM problem.

6.2. Future work

There are several avenues for potential future work. First, we could seek to speed up the algorithm by using additional data structures to keep the second-nearest facility $F_i^{(2)}$ to client $i \in I$ and the set $C_j^{(2)}$ of clients whose second-nearest facility is $j \in J$.

Second, there is scope for more consideration of the algorithm parameters γ and τ. In the comparisons above these were fixed to 0.6 and 6. Because their values can significantly affect the performance of the algorithm, a valuable future work can be the investigation of various mechanisms to set these parameters. That is, we go beyond simply tuning them for a given dataset as in Section 5.2 and set them on a per-instance basis and even change them dynamically as the algorithm runs. Machine learning may be used for this purpose. Another approach is to view the problem of obtaining the best values of γ and τ as an optimisation problem on its own and apply a high-level metaheuristic algorithm to obtain suitable values. It could also be worth investigating alternative mechanisms to set the control variable α and the reconstruction size *radius*.

Because of the success of the proposed algorithm IGV, compared to the stateof the art for OpM and the similarity of pM to OpM, another future work is to adapt the algorithm to address pM. The only difference between these problems is that the objective function is to be minimised for pM instead of maximised. Therefore, the IGV algorithm can be readily used for pM after making the following minor changes:

1. Replace '>' with '<' in lines 13 and 16 of **IG1** and **IG2** (Algorithm 2), and replace 'min' with 'max' in lines 20.
2. Change the direction of the comparison in line 5 of both the **Close_facility** and **Open_facility** functions (Algorithms 3 and 4). Change '-1' to '∞' in lines 2 and 11 in **Close_facility** and '∞' to '-1' in the same lines in **Open_facility**.

This means that another potential contribution of this paper could be to bridge the gap between the literatures of these two problems, allowing to unify the research for them. Currently, there are different algorithms and even different benchmarks in the literature of these problems.

Finally, a natural avenue for future work is to adapt the proposed algorithm to address other facility location problems. Because of its significant results in this paper, the algorithm or its ideas may even be adapted for other NP-hard optimisation problems.

Acknowledgments

The authors would like to thank Dr. O. Gökalp and Dr. J. Chang for promptly replying to enquiries and providing their source codes. They also thank the anonymous reviewers of both this and the previous version of this manuscript for their useful suggestions.

References

Beasley, J.E., 1990a. OR-Library: distributing test problems by electronic mail. *Journal of the Operational Research Society* 41, 11, 1069–1072.

Beasley, J. E., 1990b. OR-library. Available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/pmedinfo.html (accessed 30 September 2020).

Belotti, P., Labbé, M., Maffioli, F., Ndiaye, M.M., 2007. A branch-and-cut method for the obnoxious p-median problem. *4OR* 5, 4, 299–314.

Blum, C., Roli, A., 2003. Metaheuristics in combinatorial optimization: overview and conceptual comparison. *ACM Computing Surveys (CSUR)* 35, 3, 268–308.

Chang, J., Wang, L., Hao, J.K., Wang, Y., 2021. Parallel iterative solution-based tabu search for the obnoxious p-median problem. *Computers & Operations Research* 127, 105155.

Church, R.L., Drezner, Z., 2022. Review of obnoxious facilities location problems. *Computers & Operations Research* 138, 105468.

Church, R.L., Garfinkel, R.S., 1978. Locating an obnoxious facility on a network. *Transportation Science* 12, 2, 107–118.

Colmenar, J.M., Greistorfer, P., Martí, R., Duarte, A., 2016a. Advanced greedy randomized adaptive search procedure for the obnoxious p-median problem. *European Journal of Operational Research* 252, 2, 432–442.

Colmenar, J. M., Greistorfer, P., Martí, R., Duarte A., 2016b. Optsicom project, University of Valencia, Spain. Available at http://grafo.etsii.urjc.es/optsicom/opm/ (accessed 26 October 2020).

Demir, Y., 2022. An iterated greedy algorithm for the planning of yarn-dyeing boilers. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.13232

Erkut, E., Neuman, S., 1989. Analytical models for locating undesirable facilities. *European Journal of Operational Research* 40, 3, 275–291.

Gökalp, O., 2020. An iterated greedy algorithm for the obnoxious p-median problem. *Engineering Applications of Artificial Intelligence* 92, 103674.

Herrán, A., Colmenar, J.M., Martí, R., Duarte, A., 2020. A parallel variable neighborhood search approach for the obnoxious p-median problem. *International Transactions in Operational Research* 27, 1, 336–360.

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

Mladenović, N., Labbé, M., Hansen, P., 2003. Solving the p-center problem with tabu search and variable neighborhood search. *Networks: An International Journal* 42, 1, 48–64.

Mladenović, N., Brimberg, J., Hansen, P., Moreno-Pérez, J.A., 2007. The p-median problem: a survey of metaheuristic approaches. *European Journal of Operational Research* 179, 3, 927–939.

Mladenović, N., Todosijević, R., Urošević, D., 2016. Less is more: basic variable neighborhood search for minimum differential dispersion problem. *Information Sciences* 326, 160–171.

Mladenović, N., Alkandari, A., Pei, J., Todosijević, R., Pardalos, P.M., 2020. Less is more approach: basic variable neighborhood search for the obnoxious p-median problem. *International Transactions in Operational Research* 27, 1, 480–493.

Mousavi, S.R., 2023. Exploiting flat subspaces in local search for p–Center problem and two fault–tolerant variants. *Computers & Operations Research* 149, P. 106023.

Plastria, F., 1996. Optimal location of undesirable facilities: a selective overview. *JORBEL-Belgian Journal of Operations Research, Statistics, and Computer Science* 36, 2-3, 109–127.

Pullan, W., 2008. A memetic genetic algorithm for the vertex p-center problem. *Evolutionary Computation* 16, 3, 417–436.

Reese, J., 2006. Solution methods for the p-median problem: an annotated bibliography. *NETWORKS: An International Journal* 48, 3, 125–142.

Rocha, C., Pessoa, B.J., Aloise, D., Cabral, L.A., 2022. An efficient implementation of a VNS heuristic for the weighted fair sequences problem. *International Transactions in Operational Research*. https://doi.org/10.1111/itor.13197

Tamir, A., 1991. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics* 4, 4, 550–567.