Computers & Operations Research 62 (2015) 133–144

Contents lists available at ScienceDirect

Computers & Operations Research

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

Improving the quality of heuristic solutions for the capacitated vertex *p*-center problem through iterated greedy local search with variable neighborhood descent

Dagoberto R. Quevedo-Orozco, Roger Z. Ríos-Mercado *

Universidad Autónoma de Nuevo León, Graduate Program in Systems Engineering, San Nicolás de los Garza, NL 66450, Mexico

ARTICLE INFO

Available online 21 January 2015

Keywords:
Combinatorial optimization
Discrete location
Capacitated *p*-center problem
Metaheuristics
Iterated greedy local search
Variable neighborhood descent

ABSTRACT

The capacitated vertex *p*-center problem is a location problem that consists of placing *p* facilities and assigning customers to each of these facilities so as to minimize the largest distance between any customer and its assigned facility, subject to demand capacity constraints for each facility. In this work, a metaheuristic for this location problem that integrates several components such as greedy randomized construction with adaptive probabilistic sampling and iterated greedy local search with variable neighborhood descent is presented. Empirical evidence over a widely used set of benchmark data sets on location literature reveals the positive impact of each of the developed components. Furthermore, it is found empirically that the proposed heuristic outperforms the best existing heuristic for this problem in terms of solution quality, running time, and reliability on finding feasible solutions for hard instances.

1. Introduction

In this paper, the capacitated vertex *p*-center problem (C*p*CP) is addressed. This is a location problem that can be defined as follows. Given a set of nodes V representing customers and potential facility location points, where each node $j \in V$ has a demand w_j and a facility capacity s_j, a distance d_{ij} between each pair of nodes $i, j \in V$, and a given number p of desired facilities, we must decide where to locate p facilities and how to assign all the customers to these facilities so as to minimize the largest distance between any customer and its assigned facility subject to demand capacity constraints. The C*p*CP is NP-hard [1]. Several practical applications can be modeled as a C*p*CP, particularly problems that arise on emergency situations such as providing emergency medical service by locating emergency facilities or ambulances, or the location of fire stations.

Much has been devoted to the uncapacitated version of this problem. Elloumi et al. [2] provide an extensive review of the literature. However, the research on the C*p*CP has been more limited. The most significant contributions from the exact optimization perspective are due to Özsoy and Pınar [3] and Albareda-Sambola et al. [4]. In the former, the authors presented an exact method based on solving a series of set covering problems using an off-the-shelf mixed-integer linear programming (MILP) solver while carrying out an iterative search over the coverage distances. In the latter, the authors proposed an exact method based on Lagrangian relaxation and a covering reformulation.

To the best of our knowledge, the most significant heuristic method for the C*p*CP is due to Scaparra et al. [5]. In their work, the authors developed a heuristic based on large-scale local search with a multiexchange neighborhood represented by an improved graph, exploiting principles from network optimization theory. They address instances of up to 402 nodes and 40 facilities.

The main purpose of our work is to integrate several of the advanced heuristic components that have been very successful in other combinatorial optimization problems into a metaheuristic algorithm for the C*p*CP. To this end, a greedy construction heuristic with adaptive probabilistic sampling is developed. In addition, this is enhanced by a local search phase using iterated greedy local search with variable neighborhood descent. In each of these components, the particular problem structure is adequately exploited. Our empirical work indicates the positive impact of each of these components when integrated into the metaheuristic. When compared to the existing work, the proposed heuristic provides solutions of better quality than those found by the other heuristic under similar, and in many cases significantly less, computational effort. Furthermore, when compared to existing work, our heuristic is found more robust on finding feasible solutions for the hardest data sets.

The rest of the paper is organized as follows. The combinatorial optimization model for the C*p*CP is given in Section 2. In Section 3, a detailed description of the proposed algorithm is presented. Section 4

* Corresponding author. Tel.: +52 81 8329 4000x1634.
E-mail addresses: dago@yalma.fime.uanl.mx (D.R. Quevedo-Orozco), roger.rios@uanl.edu.mx (R.Z. Ríos-Mercado).

http://dx.doi.org/10.1016/j.cor.2014.12.013
0305-0548/

shows the empirical results, providing a description of the benchmark data sets, parameter fine-tuning, comparison among methods, and component analysis of the proposed heuristic. Concluding remarks are given in Section 5.

2. Problem formulation

Let $U=\{u_1,\ldots,u_m\}$ be the set of potential locations for the p facilities and $V=\{v_1,\ldots,v_n\}$ be the set of customers, this paper assumes the special case when $U=V$. Let d_{ij} denote the integer distance between nodes i and j. Each $j\in V$ has an integer demand w_j and an integer capacity s_j. A p-partition of V is given by $X=\{X_1,\ldots,X_p\}$ and $K=\{1,\ldots,p\}$, where $X_k\subset V$ is a subset of V. Each subset X_k is formed by a subset of nodes such that $\bigcup_{k\in K}X_k=V$ and $X_k\cap X_q=\varnothing$ for all $k,q\in K, k\neq q$. The set of centers is given by $P=\{c(1),\ldots,c(p)\}$, where $c(k)$ is the center for subset X_k, i.e., the node that hosts the facility serving the customers in X_k. The problem can be represented as

$$\min_{X\in\Pi}\ \max_{k\in K} f(X_k), \tag{1}$$

where Π is the collection of all p-partitions of V, and its objective function is given by $f(X_k)=\max_{j\in X_k}\{d_{j,c(k)}\}$. The center $c(k)$ is given by

$$c(k)=\arg\min_{i\in X_k}\left\{\max_{j\in X_k}\left\{d_{ij}:\sum_{j'\in X_k} w_{j'}\le s_i\right\}\right\}. \tag{2}$$

If for a given X_k there is not any $i\in X_k$ such that $\sum_{j\in X_k} w_j\le s_i$ then $f(X_k)=\infty$. Here, computing $c(k)$ comes basically from solving a 1-center problem which can be done rather quickly in $\mathcal{O}(n)$. We define by $\ell(j)$ the index $k\in K$ where customer j belongs to, i.e., $\ell(j)=k$ if and only if $j\in X_k$.

3. Description of the heuristic

In recent years, an important trend in the metaheuristic field is that of integrating different components resulting in successful hybrid methods that attempt to better exploit the specific problem structure. As a direct consequence of this, sometimes it is not clear how to name a specific heuristic as it uses ideas from several methods. In this regard, the proposed solution method uses an Iterated Greedy Local Search (IGLS) procedure as its guiding framework. In its construction phase, a greedy mechanism with adaptive biased sampling is employed. Then IGLS with Variable Neighborhood Descent (VND) is applied in the improvement phase.

IGLS is a technique originally proposed by Ruiz and Stützle [6] that extends the Iterated Local Search (ILS) heuristic [7]. IGLS iteratively applies phases of destruction, reconstruction, and local search to a given input solution. In order to escape from a local optimum and to explore other regions in the solution space, IGLS applies a perturbation procedure to generate new starting points for the local search by changing the current solution. The destruction/reconstruction or perturbation consists of removing some elements of the current solution under a specific criterion, followed by a greedy algorithm to obtain a new solution. After a solution has passed through a perturbation phase, a local search procedure is applied. The last step is to decide, based on an improvement criterion, if the solution obtained after the local search should replace the incumbent solution for the next iteration. The IGLS algorithm iterates over these steps until some stopping criterion is met. An advantage of IGLS is that it allows to diversify and improve along the search without the need of employing complex memory structures. Its simplicity makes it applicable to several combinatorial optimization problems. For instance, Ruiz and Stützle [6,8], Fanjul-Peyro and Ruiz [9] and Urlings et al. [10] provide state-of-the-art results for different scheduling problems.

VND is an iterative improvement algorithm where t neighborhoods, which are typically ordered according to increasing size, are used. The algorithm starts with the first neighborhood and performs iterative improvement steps until local optimality is reached. When no further improvement is found for the h-th neighborhood and $h+1\le t$, VND continues the search in the $(h+1)$-th neighborhood. If an improvement is found, the search process starts again at the first neighborhood. When $h+1>t$ then the search process ends and the procedure returns the final solution, which is a local optimum with respect to all t neighborhoods. It has been shown that VND can considerably improve the performance of iterative improvement algorithms with respect to both the solution quality of the final solution and the time required for finding high-quality solutions compared to using standard iterative improvement in large neighborhoods [11].

This idea of hybridizing ILS with VND has been successfully used in the past for other combinatorial optimization problems. For instance, Subramanian et al. [12] used a parallel version for the VRP with simultaneous pickup and delivery. Martins et al. [13] proposed a method for solving a routing and wavelength assignment problem for optical networks.

The proposed approach is depicted in Algorithm 1. An initial solution is obtained (Steps 2–3). The local search (Steps 6–7) is done as long as the solution keeps improving. Let $\mathcal{B}(X)$ be the set of bottleneck subsets in X, i.e., $\mathcal{B}(X)=\{k\in K: f(X_k)=f(X)\}$ and $\mathcal{J}(X)$ contains the demand nodes with maximum distance from the active center node in each subset X_k, i.e., $\mathcal{J}(X)=\{j\in X_k: d_{jc(k)}=f(X), k\in\mathcal{B}(X)\}$. To compare two solutions X and X', we use an effective improvement criterion proposed in [5] that includes the reduction of bottleneck elements, as follows. It is stated that

$$X' \text{ is better than } X \Leftrightarrow \begin{cases} f(X')<f(X) & \text{or}\\ f(X')=f(X), & \mathcal{B}(X')\subseteq\mathcal{B}(X), \mathcal{J}(X')\subset\mathcal{J}(X). \end{cases} \tag{3}$$

Algorithm 1. Pseudo-code of IGLS that takes as input: a set of nodes V, number p of desired facilities, a value $\alpha\in[0.0,1.0]$ of destruction of the solution during the perturbation step, and the maximum number of iterations $r_{\max}$.

```
1: procedure IGLS (V, p, α, r_max)
2:   X' ← Construction(V,p)                    ▷ Construction
3:   X' ← VND(X')
4:   X ← X'
5:   while r_max > 0 do                        ▷ Improvement
6:     X' ← Perturbation(X', α)
7:     X' ← VND(X')
8:     if X' is better than X then
9:       X ← X'
10:    else
11:      X' ← Shake(X')
12:    end if
13:    r_max ← r_max − 1
14:  end while
15:  return X
16: end procedure
```

This criterion is met if X' decreases the objective function value or if X' reduces the number of bottleneck customers while not worsening the total cost, without creating new bottleneck subsets and new bottleneck customers. The incumbent solution X is updated if a better feasible solution is found according to the criterion (3);

otherwise, a shake (Step 11) of the solution X' is applied. The approach stops when the maximum number of iterations (denoted by $r_{\max}$) is met. Each of the four main components (e.g., Construction(), Perturbation(), VND(), and Shake()) of Algorithm 1 is described next.

3.1. Construction

The construction phase consists of two stages: (a) center location and (b) customer allocation. In the former, we used a greedy randomized strategy with adaptive probabilistic selection, which diversifies the set of potential centers. In the latter, a deterministic greedy approach based on both a distance criterion and node capacity is performed.

3.1.1. Stage (a): Center location

The goal here is to choose an adequate set of p centers or seed nodes. The choice of these centers is made through a greedy randomized construction procedure, taking into account the distance factors and the capacity of each node $j \in V$. The location phase starts by choosing the first center i^* randomly. Then, we iteratively choose the next center seeking a node whose greedy function value, given by

$$\gamma(j) = s_j d_{i^* j}, \tag{4}$$

is relatively large. The motivation of this function is to try to obtain centers that are as disperse as possible, but also to favor the choice of centers with large capacity such that we can assign more customers to it in the allocation phase. Within a greedy randomized method this is done as follows. Let P be a partial set of chosen centers. Then, for each $j \in V \setminus P$ we compute its greedy function (4) value. Note that when the capacity of j and the distance between i^* and j increases, the value of the greedy function for j increases, making it more attractive to be selected as the next center because this node is disperse regarding the set P and its capacity is large; on the other hand, if distance or capacity values get smaller then the value of this function is decreased and j is not attractive to be a center. The probability $\pi(j)$ of selecting the next element $j \in V \setminus P$ is computed as

$$\pi(j) = \frac{\gamma(j)}{\sum_{j' \in V \setminus P} \gamma(j')}. \tag{5}$$

By using this probability measure, nodes with better greedy function evaluation have better chance of being chosen. Note also that the role of the $\pi(j)$'s is invariant to the scale of s and d in (4), thus there is no need to normalize these factors in (4). In a preliminary version of this work [14], the RCL was restricted by the usual GRASP quality threshold parameter; however, we found the new less restrictive approach much better.

3.1.2. Stage (b): Customer allocation

Once the centers are fixed, the second stage consists of allocating the customers to these centers. This stage is done in a deterministic greedy fashion, because some preliminary tests show that deterministic choices outperform randomizing strategies. The assignments are defined by the remaining nodes $j \in V \setminus P$. To this end we define a greedy function that measures the cost of assigning a customer j to a center k located in $c(k)$ as

$$\phi(j,k) = \begin{cases} \dfrac{d_{jc(k)}}{\overline{d}} & \text{if } w_j \le r(k), \\ w_j - r(k) & \text{if } w_j > r(k); \end{cases} \tag{6}$$

where $\overline{d} = \max_{i,j \in V}\{d_{ij}\} + 1$ is a normalization factor and $r(k) = s_{c(k)} - \sum_{i \in X_k} w_i$ is the residual capacity for the set whose center k is located in $c(k)$. Note that the term $d_{jc(k)}/\overline{d}$ is a fraction less than one and $r(k) - w_j$ is an integer greater or equal to one. Thus, the aim is to penalize the constraint violation highly and to consider the distance factor only when the constraint is met. Afterwards, each node j is assigned to a center k^* that minimizes function (6), that is, $X_{k^*} \leftarrow X_{k^*} \cup \{j\}$, where $k^* = \arg\min_{k \in K} \phi(j,k)$. Note that because the way (6) is defined there is no way a node j is assigned to a center with no sufficient capacity while existing other centers with sufficient capacity. Finally, once the assignment is done, the centers for the entire partition are updated using (2). Algorithm 2 depicts the construction method.

Algorithm 2. Pseudo-code of construction step that takes as input: a set of nodes V and a number p of desired facilities.

```
1:  procedure Construction(V, p)
2:    P ← ∅                                              ▷ Stage (a)
3:    Choose i* ∈ V randomly
4:    P ← P ∪ {i*}
5:    Update γ(j) and π(j), j ∈ V \ P
6:    while |P| < p do
7:      Choose i* ∈ V \ P randomly using the probability π(j)
8:      P ← P ∪ {i*}
9:      Update γ(j) and π(j), j ∈ V \ P
10:   end while
11:   X ← ∅                                              ▷ Stage (b)
12:   for all j ∈ V \ P do
13:     k* ← arg min_{k∈K} φ(j,k)
14:     X_{k*} ← X_{k*} ∪ {j}
15:     Update c(k*)
16:   end for
17:   return X
18: end procedure
```

It should be noted that this phase does not guarantee construction of a feasible solution because during the allocation stage the capacity constraints may be violated for some customers. However, the local search phase successfully handles this issue as it is empirically shown in Section 4.

3.2. Local search

To attempt to improve the solution, a local search phase that uses VND within an IGLS framework is performed (Steps 5–14 in Algorithm 1). In this phase a perturbation consisting of unassigning and reassigning customers to centers according to the IGLS idea is done first (Step 6). Then, this is followed by VND (Step 7), where two different neighborhood structures are used. Finally, if the current solution is no better than the incumbent, a more aggressive destruction/reconstruction procedure is performed (Step 11). Each of these components is explained in detail next.

3.2.1. Perturbation

This method takes a solution as an input and applies destruction and reconstruction steps. The objective of this procedure is to reduce bottleneck elements using specific criteria of node unassignment for each subset (Steps 3–10). In this specific case, for a chosen X_k, the destruction step is done by unassigning the α% of nodes located in X_k, with high values of the probability function defined by

$$\rho(j) = \frac{d_{jc(k)}}{\sum_{j' \in X_k \setminus \{c(k)\}} d_{j'c(k)}}. \tag{7}$$

The choice of this function is motivated by the fact that the nodes farther from the center are the ones affecting more the dispersion function value.

Algorithm 3. Pseudo-code of perturbation step that takes as input: a solution X and a value $\alpha \in [0.0, 1.0]$ that represent the percent of nodes to be disconnected from a solution.

1: **procedure** PERTURBATION(X, α)
2: $W \leftarrow \emptyset$ ▷ Destruction
3: **for all** $k \in K$ **do**
4: $Q \leftarrow X_k \setminus \{c(k)\}$
5: Update $\rho(j), j \in Q$
6: **while** $\frac{|W|}{|X_k|-1} < \alpha$ **do**
7: Choose $j \in Q$ randomly using probability $\rho(j)$
8: $W \leftarrow W \cup \{j\}$
9: **end while**
10: **end for**
11: Sort W from worst to best $d_{jc(\ell(j))}, j \in W$
12: $X \leftarrow X \setminus W$
13: **for all** $j' \in W$ **do** ▷ Reconstruction
14: $k^* \leftarrow \arg\min_{k \in K} \phi(j', k)$
15: $X_{k^*} \leftarrow X_{k^*} \cup \{j'\}$
16: **end for**
17: Update $c(k), k \in K$
18: **return** X
19: **end procedure**

The reconstruction step (Steps 13–16) reassigns each unassigned node to a center k^* that minimizes function (6). A priority assignment is given to the bottleneck nodes, i.e., nodes whose previous assignment matched the value of the objective function value; this is achieved through a sort on the set of unassigned nodes from worst to best previous distance (Step 11). Finally, once the reconstruction is done, the centers for the entire partition are updated (Step 17) using (2). The pseudo-code of this perturbation method is shown in Algorithm 3.

It is important to point out that an improvement in the objective function value is not necessarily achieved by the perturbation method. This is due to the fact that there might be some nodes that can worsen the objective value of the current solution during the reconstruction step. However; this is not precisely bad news because it allows more diversification to a given solution which can later be improved by VND in the following step. As it is usual in IGLS, the value of the destruction parameter α plays an important role in the quality of the solutions found by the method. Appropriate values are empirically investigated and fine-tuned for a large number of instances in the benchmark data sets as shown in Section 4.2.

3.2.2. VND

Our VND implementation, depicted in Algorithm 4, employs two different neighborhood structures denoted by $\mathcal{N}_1$ and $\mathcal{N}_2$.

For each $i \in \mathcal{J}(X)$ we applied the best move strategy of the h-th neighborhood (Steps 4–6). For each of the two neighborhoods, the potential move takes into account both distance and capacity factors. Given the nature of the objective function, it makes perfect sense to consider only a subset of promising moves, not the entire neighborhood, i.e., we only consider moves where the distance between i to potential target subset $c(k)$ is strictly less than $f(X)$ and the capacity is not exceeded in any center involved in the movement. These restrictions rule out infeasible movements, reducing the size of the neighborhood to be explored, and increasing the performance of the procedure. Therefore, a best improving move strategy is adopted through the search. Once the move is made, the centers for the entire partition are updated using (2). Finally, the incumbent solution X is updated if the solution X' is best according to criterion (3). Each neighborhood is described next.

($\mathcal{N}_1$) *Reinsertion*: This neighborhood considers moves where a node $i \in \mathcal{J}(X)$ (currently assigned to set X_k) is reassigned to set X_q, with $k \neq q$, i.e., given a p-partition X, the move is defined by $reinsertion(i, q) = \{X_1, \ldots, X_k \setminus \{i\}, \ldots, X_q \cup \{i\}, \ldots, X_p\}$. The move is valid if and only if it does not exceed the capacity of the target subset X_q and $d_{ic(q)} < f(X)$. The evaluation of each feasible move is performed by using the function:

$$\psi_1(i, q) = d_{ic(k)} - d_{ic(q)}. \quad (8)$$

This function obtains high values when the distance between i and $c(q)$ is less than the current assignment i to $c(k)$. Therefore high values of $\psi_1(i, q)$ are preferred.

($\mathcal{N}_2$) *Exchange*: This neighborhood considers moves where a node $i \in \mathcal{J}(X)$ (currently assigned to set X_k) and a node $j \in X_q \setminus \{c(q)\}$, with $k \neq q$, are swapped, i.e., given a p-partition X, the move is defined by $\text{swap}(i, j) = \{X_1, \ldots, X_k \cup \{j\} \setminus \{i\}, \ldots, X_q \cup \{i\} \setminus \{j\}, \ldots, X_p\}$. The move is valid if and only if it does not exceed the capacity of the respective target subsets and $d_{jc(k)} < f(X)$ and $d_{ic(q)} < f(X)$. The evaluation of each feasible move is performed by using the function:

$$\psi_2(i, j) = (d_{ic(k)} - d_{ic(q)}) + (d_{jc(q)} - d_{jc(k)}). \quad (9)$$

This function obtains high values when the distance between i and $c(q)$ is less than the current assignment i to $c(k)$, and the distance between j and $c(k)$ is less than the current assignment j to $c(q)$. Therefore high values of $\psi_2(i, j)$ are preferred.

Algorithm 4. Pseudo-code of VND method that takes as input a solution X.

1: **procedure** VND(X)
2: **while** $h \leq 2$ **do**
3: $X' \leftarrow X$
4: **for all** $i \in \mathcal{J}(X)$ **do**
5: $X' \leftarrow \text{best}(\mathcal{N}_h(i, X'))$
6: **end for**
7: **if** X' is better than X **then**
8: $X \leftarrow X'$
9: $h \leftarrow 1$
10: **else**
11: $h \leftarrow h + 1$
12: **end if**
13: **end while**
14: **return** X
15: **end procedure**

3.2.3. Shake

The previously described perturbation and VND methods consider moves where centers tend not to change too much. The computational effort of every move evaluation is reasonable. However, it is possible to force a more aggressive destruction/reconstruction mechanism that would cause centers to change considerably. This is achieved by a "shake" mechanism that is applied after a local optima is reached under the iterative application of the perturbation and VND. This, of course, requires more computational effort but it pays off. The shake method performs an iterative removal and reconstruction of several subsets, which greatly diversifies the search path.

The procedure (depicted in Algorithm 5) works as follows. Given a p-partition $X = (X_1, \ldots, X_p)$, and a bottleneck node $i \in \mathcal{J}(X)$, we define the set $W^{(i)} = \{X_{k_1}, \ldots, X_{k_q}\}$ of q subsets with $\{k_1, \ldots, k_q\} \subset K$, whose corresponding centers are the q nearest centers to i. This is an ordered set, that is, $c(k_1)$ is the closest center to i. Note that k_1 may not necessarily be equal to $\ell(i)$, i.e., node i may not be assigned to its closest center. Now, for each bottleneck node $i \in \mathcal{J}(X)$, such that

$k_1 = \ell(i)$, these q subsets in $W^{(i)}$ are to be destroyed, that is, each node $i \in W^{(i)}$ is unassigned from its current set. The reason for this is that it makes more sense to destroy territories where bottleneck nodes are assigned to its closest center as it is more likely to change indeed the value of the min–max objective function. Parameter q is fixed at $q = \lceil \ln p \rceil + 1$ which works reasonably well in preliminary testing. The nodes in W are then unassigned or removed from X (Step 5). Then, we construct a new partial q-partition X' (Step 6) that allows to obtain a new solution X (Step 7).

Algorithm 5. Pseudo-code of the Shake method that takes as input a solution X.

```
1: procedure SHAKE(X)
2:   q ← ⌈ln p⌉ + 1
3:   for all i ∈ 𝒯(X) such that k_1 = ℓ(i) do
4:     Set the q nearest subsets to i: W^(i) ← {X_{k_1}, ..., X_{k_q}}
5:     X ← X \ W^(i)
6:     X' ← CONSTRUCTION(W, q)
7:     X ← X ∪ X'
8:   end for
9:   return X
10: end procedure
```

This reconstruction step introduces a randomized mechanism, influenced by the construction phase. This shake mechanism does not guarantee neither reducing bottleneck nodes nor improving the objective value; however, the aggressive destruction diversifies the partial structure of the current solution, selecting other subsets of potential centers which may be desirable to guide the method to other most promising search regions during the improvement phase to be performed later.

Fig. 1 shows a segment of a solution which is eligible for applying the shake method. In this example, node i is a bottleneck node currently assigned to its closest center k, i.e., $k = \ell(i)$. Therefore, under the selection criterion previously defined with $q = 3$, the region formed by the subsets with centers k, l, and m is destroyed. Then these three sets are rebuilt using the construction step defined in Algorithm 2, yielding a new solution.

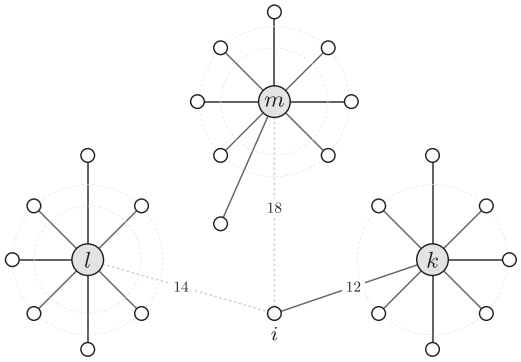

Fig. 1. Segment of a solution which is eligible for applying the shake method.

4. Computational results

The proposed heuristic is evaluated on benchmark data sets from the literature. The heuristic was implemented in C++ and compiled with GNU g++ version 4.4.5. To obtain exact solutions or lower bounds, the exact methods of Özsoy and Pınar (OP) [3] and Albareda-Sambola et al. (ADF) [4] were used. In these methods, ILOG CPLEX 12.5 is used as LP solver. A time limit of 1 h and a memory usage limit of 1 Gb for data sets A, B, C, and D-*, and 6 h and a memory usage limit of 4 Gb for data set E were set as stopping criteria. Each of the experiments was carried out on a platform with AMD Opteron 2.0 GHz (x16), 32 GiB RAM under Debian 6.0.8 GNU/Linux Kernel 2.6.32-5, 64 bit architecture.

4.1. Description of test bed

For the experiments, we used five different data sets. No benchmark instance data sets for the C*p*CP exist in the literature; however, we tested and compared all methods using data sets generated for other location problems and used in previous work on this problem.

(Set A) Beasley OR-Library: This data set, proposed in [15] for the capacitated p-median problem, contains two groups of 10 instances with equal facility capacity. One has 50 demand nodes and 5 facilities to be located, and the other has 100 demand nodes and 10 facilities to be located.

(Set B) Galvão and ReVelle: This data set was generated by Scaparra et al. [5] specifically for the C*p*CP based on the data set of Galvão and ReVelle for the maximal covering location problem [16]. The data sets contain instances with 100 and 150 customers, and 5–15 centers, with variable facility capacity. The original set is composed of two networks that were randomly generated.

(Set C) Lorena and Senne: This set, proposed in [17] for the capacitated p-median problem (C*p*MP), includes 6 large instances whose sizes ranges from 100 to 402 customers, and from 10 to 40 centers, and equal facility capacity.

(Set D) Ceselli and Righini: There exists a recent data set added to the OR-Library proposed in [18]. This is regarded as a very hard set to solve for capacitated location problems such as the p-median problem, and may be significantly influenced by the ratio between n and p. The lower the n/p ratio is the more difficult the instance tends to be. We have observed a similar behavior when using this set as data for our capacitated p center problem. In the previous sets A, B, and C, this n/p ratio is 10 or more. This D set is composed of 4 subsets α, β, γ and δ, each subset consists of forty instances created from the following set based: 20 of them have a graph with 50 and 100 nodes; the other 20 instances were randomly generated on graphs with cardinality 150 and 200. Therefore, the same 40 instances were solved with different number of centers for each data set: α with $p = \lfloor n/10 \rfloor$, β with $p = \lfloor n/4 \rfloor$, γ with $p = \lfloor n/3 \rfloor$ and δ with $p = \lfloor 2n/5 \rfloor$. The overall capacity was preserved in all subsets by setting $s_i = \lceil 12n/p \rceil$. We refer to these four different data sets as D-α, D-β, D-γ, and D-δ, whose literal expresses the complexity of each subset, where α is the easy subset and δ is the complex subset.

(Set E) Reinelt: This set was generated by Lorena and Senne [17] specifically for the C*p*MP based on the data set of instance in the TSPLIB compiled by Reinelt [19]. The data set contains 5 instances with 3038 customers and 600–1000 centers and equal facility capacity for instance, calculated as $s_i = \lceil \sum w_j/p \times 0.8 \rceil$. This set is considered large scale and ratios n/p between 3 and 5.

4.2. Fine-tuning

The purpose of this first experiment is to fine-tune the heuristic with respect to the destruction parameter α for each data set. This is achieved by a detailed statistical analysis as described below.

The response variable studied is the average relative percentage deviation or gap as follows:

$$\text{GAP} = \frac{f(X) - f^{lb}}{f^{lb}} \times 100, \tag{10}$$

where $f(X)$ is the resulting objective function value found for each particular instance and f^{lb} its best known lower bound. This bound was obtained by applying either of the exact methods OP or ADF. In some cases, this computed lower bound turned out to be the optimal solution, but when an exact method met the stopping condition by time limit or memory usage, a lower bound is given. The heuristic iteration limit was set to 100 for data sets A, B, C, and D-*, and 50 for data set E; and was run for each value $\alpha \in \{0.0, 0.1, \ldots, 1.0\}$ on all data sets, performing 5 replicates in this experiment. We separate the results into eight blocks, according to the number of data sets with subsets and study a single factor α of each block composed by 5 replicates, where GAP is the response variable.

Based on test with Tukey's Honest Significant Difference (HSD) 95%, it was observed that the set {0.7,0.6,0.7,0.6,0.4,0.5,0.4,0.4} of α values gave the best results for data sets A, B, C, D-α, D-β, D-γ, D-δ, and E, respectively. For the remaining experiments, we used these selected values for α and run our heuristic using 1000 as iteration limit for data sets A, B, C, and D-*, and 200 as iteration limit for data set E, with 30 repetitions.

4.3. Comparison between heuristics

The main purpose of these experiments is to provide a detailed comparison between the proposed heuristic (QR) and the one by Scaparra et al. (SPS) [5]. Version 1S with exact recentering, based on the stack implementation of the set of candidates nodes Q, and restricted relocation of the SPS heuristic was used. In addition, we have decided to include the results obtained by the exact methods OP and ADF.

Tables 1–8 display the comparison of methods for each data set. In each table the first two columns represent the instance size measured by number of nodes n and number of partitions p. "Instance" is the name of the particular problem instance and "Optimal" indicates the optimal value of the instance or the best known lower bound denoted by "*" beside the value. The section "Time (s)" gives the execution time in seconds and "Deviation (%)" expresses the percent of relative deviation or gap with respect to

Table 1
Comparison of methods on data set A.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR[1]	QR[2]
50	5	cpmp01	29	0.07	0.32	0.71	0.22	0.00	0.00	0.00	0.00	0.00
		cpmp02	33	0.13	2.05	0.97	0.20	0.00	0.00	0.00	0.00	0.00
		cpmp03	26	0.29	0.47	0.91	0.22	0.00	0.00	7.69	0.00	0.00
		cpmp04	32	0.09	1.00	1.01	0.21	0.00	0.00	0.00	0.00	0.00
		cpmp05	29	2.22	1.79	1.11	0.23	0.00	0.00	0.00	0.19	0.00
		cpmp06	31	2.19	2.72	1.06	0.22	0.00	0.00	3.23	3.07	0.00
		cpmp07	30	0.16	0.82	1.23	0.23	0.00	0.00	0.00	0.80	0.00
		cpmp08	31	0.18	1.10	1.09	0.24	0.00	0.00	0.00	1.29	0.00
		cpmp09	28	3.46	7.22	1.29	0.23	0.00	0.00	0.00	3.25	0.00
		cpmp10	32	4.41	6.39	2.50	0.23	0.00	0.00	9.38	12.81	0.00
		Average		1.32	2.39	1.19	0.22	0.00	0.00	2.03	2.14	0.00
100	10	cpmp11	19	6.38	5.48	9.34	0.58	0.00	0.00	5.26	9.62	0.00
		cpmp12	20	0.33	5.98	10.21	0.52	0.00	0.00	15.00	5.22	0.00
		cpmp13	20	11.27	6.54	9.32	0.54	0.00	0.00	10.00	0.64	0.00
		cpmp14	20	2.40	4.96	8.70	0.54	0.00	0.00	10.00	2.44	0.00
		cpmp15	21	2.66	7.28	9.64	0.56	0.00	0.00	9.52	3.73	0.00
		cpmp16	20	22.84	12.38	10.35	0.56	0.00	0.00	5.00	4.42	0.00
		cpmp17	22	78.51	553.31	9.91	0.57	0.00	0.00	4.55	4.70	4.55
		cpmp18	21	10.14	9.77	8.21	0.56	0.00	0.00	9.52	1.77	0.00
		cpmp19	21	5.00	19.06	10.16	0.58	0.00	0.00	9.52	5.66	0.00
		cpmp20	21	397.88	20.55	8.97	0.61	0.00	0.00	9.52	9.20	0.00
		Average		53.74	64.53	9.48	0.56	0.00	0.00	8.79	4.74	0.46
		Overall average		27.53	33.46	5.34	0.39	0.00	0.00	5.41	3.44	0.23

Table 2
Comparison of methods on data set B.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR[1]	QR[2]
100	5	G1	94	159.94	9.02	7.00	0.33	0.00	0.00	2.13	1.80	1.06
100	5	G2	94	53.37	12.41	7.09	0.33	0.00	0.00	2.13	1.83	0.00
100	10	G3	83	55.19	122.17	14.02	0.67	0.00	0.00	8.43	8.68	4.82
100	10	G4	84	113.05	41.76	15.21	0.74	0.00	0.00	7.14	8.47	5.95
150	10	G5	95	430.61	302.30	40.55	0.90	0.00	0.00	7.37	4.45	2.11
150	10	G6	96	155.52	224.46	36.85	0.88	0.00	0.00	7.29	4.23	2.08
150	15	G7	89	208.94	102.64	49.78	1.16	0.00	0.00	8.99	8.00	5.62
150	15	G8	89	251.59	421.37	47.88	1.07	0.00	0.00	10.11	8.77	5.62
		Overall average		178.53	154.52	27.30	0.76	0.00	0.00	6.70	5.78	3.41

the optimal value or best known lower bound for each method. In the case of exact methods, the gap represents the deviation between best lower and upper bound found. For the proposed method QR, we show the average time performance over the 30 independent repetitions, also "$QR^1\%$" and "$QR^2\%$" denote the average and best GAP, respectively, over all repetitions. Table 9 summarizes the comparison among methods for all data sets in terms of their average of the percent of relative deviation, running

Table 3
Comparison of methods on data set C.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR^1	QR^2
100	10	SJC1	364	262.10	368.97	14.59	0.54	0.00	0.00	26.67	25.52	5.28
200	15	SJC2	304	58.24	114.24	79.90	1.09	0.00	0.00	19.02	5.88	1.60
300	25	SJC3a	278	142.70	283.95	234.15	2.31	0.00	0.00	36.94	8.97	3.24
300	30	SJC3b	253	111.08	277.79	247.68	2.66	0.00	0.00	30.86	8.23	3.27
402	30	SJC4a	284	576.16	1100.08	631.14	3.37	0.00	0.00	42.29	8.58	4.51
402	40	SJC4b	239	251.51	343.50	505.12	4.36	0.00	0.00	34.93	9.15	3.60
		Overall average		233.63	414.76	285.43	2.39	0.00	0.00	31.79	11.06	3.58

Table 4
Comparison of methods on data set D-α.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR^1	QR^2
50	5	01	29	0.07	0.33	0.70	0.22	0.00	0.00	0.00	0.00	0.00
		02	33	0.12	2.04	0.98	0.20	0.00	0.00	0.00	0.00	0.00
		03	26	0.30	0.46	0.90	0.22	0.00	0.00	7.69	0.00	0.00
		04	32	0.09	1.01	1.01	0.21	0.00	0.00	0.00	0.00	0.00
		05	29	2.26	1.81	1.11	0.23	0.00	0.00	0.00	1.04	0.00
		06	31	2.17	2.72	1.05	0.22	0.00	0.00	3.23	3.18	0.00
		07	30	0.16	0.81	1.22	0.23	0.00	0.00	0.00	0.93	0.00
		08	31	0.19	1.07	1.08	0.24	0.00	0.00	0.00	2.49	0.00
		09	28	2.89	8.37	1.30	0.23	0.00	0.00	0.00	3.50	0.00
		10	32	4.27	6.88	2.51	0.22	0.00	0.00	9.38	12.75	0.00
		Average		1.25	2.55	1.19	0.22	0.00	0.00	2.03	2.39	0.00
100	10	11	19	6.31	5.52	9.18	0.56	0.00	0.00	5.26	11.37	0.00
		12	20	0.34	5.75	10.21	0.54	0.00	0.00	15.00	5.30	0.00
		13	20	11.12	6.65	9.67	0.54	0.00	0.00	10.00	1.75	0.00
		14	20	2.45	4.70	8.68	0.54	0.00	0.00	10.00	3.40	0.00
		15	21	2.68	7.28	9.60	0.55	0.00	0.00	9.52	4.19	0.00
		16	20	23.69	12.34	10.34	0.55	0.00	0.00	5.00	4.80	0.00
		17	22	77.74	554.23	9.78	0.57	0.00	0.00	4.55	5.86	4.55
		18	21	10.16	9.86	8.21	0.56	0.00	0.00	9.52	2.38	0.00
		19	21	4.94	19.84	10.20	0.56	0.00	0.00	9.52	6.85	0.00
		20	21	395.24	20.38	8.87	0.59	0.00	0.00	9.52	9.52	9.52
		Average		53.47	64.65	9.48	0.56	0.00	0.00	8.79	5.54	1.41
150	15	21	16	29.86	31.96	27.48	0.98	0.00	0.00	25.00	11.25	6.25
		22	17	103.79	1386.81	24.76	1.01	0.00	0.00	11.76	4.83	0.00
		23	16	33.12	37.78	23.35	1.03	0.00	0.00	18.75	11.88	0.00
		24	16	54.09	56.31	29.18	1.01	0.00	0.00	25.00	12.63	6.25
		25	16	0.72	32.71	22.99	0.95	0.00	0.00	12.50	0.00	0.00
		26	16	14.66	57.28	28.48	0.97	0.00	0.00	12.50	8.88	6.25
		27	18	3303.97	130.39	27.22	1.08	0.00	0.00	11.11	10.28	5.56
		28	17	21.43	48.93	28.06	0.96	0.00	0.00	5.88	0.00	0.00
		29	15	18.68	40.25	25.69	0.94	0.00	0.00	20.00	13.06	6.67
		30	15	4.58	52.53	28.87	0.96	0.00	0.00	20.00	6.80	6.67
		Average		358.49	187.50	26.61	0.99	0.00	0.00	16.25	7.96	3.77
200	20	31	14	45.00	29.83	52.55	1.50	0.00	0.00	21.43	5.57	0.00
		32	14	3600.75	1798.05	86.42	1.80	7.14	0.00	28.57	27.94	14.29
		33	14	58.70	101.90	57.16	1.43	0.00	0.00	28.57	11.07	7.14
		34	15	974.15	168.16	60.03	1.70	0.00	0.00	26.67	12.81	6.67
		35	14	94.71	83.85	66.88	1.58	0.00	0.00	14.29	10.91	7.14
		36	14	48.37	46.66	57.54	1.53	0.00	0.00	14.29	10.00	7.14
		37	14	8.03	89.52	65.99	1.55	0.00	0.00	28.57	7.00	0.00
		38	14	710.26	321.27	66.66	1.63	0.00	0.00	28.57	18.18	7.14
		39	13	66.72	215.80	56.29	1.48	0.00	0.00	23.08	12.77	7.69
		40	15	475.44	186.77	66.56	1.57	0.00	0.00	20.00	6.54	0.00
		Average		608.21	304.18	63.61	1.58	0.71	0.00	23.40	12.28	5.72
		Overall Average		255.36	139.72	25.22	0.84	0.18	0.00	12.62	7.04	2.72

time, and the number of instances for each method that cannot find any feasible solutions before the stopping criteria were met (this event is indicated by "–" in the column).

As far as data set A is concerned (Table 1), the exact methods were found very efficient for the smaller instance group (size 50×5), performing better than any heuristic. However, when attempting the larger group (size 100×10), there are a couple of instances for which the exact method struggled. The performance of both heuristics was more robust than that of the exact method as they both took less than 5.34 s to solve each instance. In terms of solution quality, the proposed heuristic found better solutions than the ones reported by the SPS heuristic.

When analyzing data set B (Table 2) we can observe that the exact methods take considerably longer than both heuristics to reach an optimal solution. In terms of heuristic solution quality, our heuristic obtains slightly better solutions (average GAP of 5.78%) than the SPS heuristic (average GAP of 6.70%).

Regarding data set C (Table 3), we can observe that the best exact method takes on average near 4 min while our heuristic takes less than 3 s. When comparing our heuristic with the SPS heuristic, we can see that ours is faster and finds solutions of significantly better quality.

The results of data sets D-* are displayed in Tables 4–7. As discussed in previous sections, this set is considered very difficult because their value of p has a significant effect on the performance of the methods. When analyzing subset D-α (Table 4), we observed our heuristic outperforms heuristic SPS in terms of both solution quality and computational effort. Regarding subset D-β (Table 5), SPS fails in finding feasible solutions to a significant number of instances while our heuristic provides feasible solutions for all instances. Finally, regarding subsets D-γ and D-δ (Tables 6 and 7) the infeasibility level of SPS increases considerably, while QR provides a feasible solution for all instances of subset D-γ and all but two instances in D-δ.

Table 5
Comparison of methods on data set D-β.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR[1]	QR[2]
50	12	01	19	1.20	32.27	1.18	0.43	0.00	0.00	0.00	5.16	0.00
		02	20	0.93	3.07	2.19	0.42	0.00	0.00	10.00	13.70	10.00
		03	20	0.85	39.52	1.96	0.47	0.00	0.00	10.00	11.40	5.00
		04	24	7.92	5.35	2.51	0.49	0.00	0.00	8.33	20.75	16.67
		05	21	46.47	15.40	3.28	0.45	0.00	0.00	9.52	21.48	14.29
		06	22	8.60	5.11	3.73	0.41	0.00	0.00	4.55	16.00	0.00
		07	20	2.32	2.10	4.99	0.41	0.00	0.00	0.00	10.39	0.00
		08	23	2.28	2.63	3.78	0.39	0.00	0.00	4.35	4.35	4.35
		09	19	7.60	14.65	0.45	0.35	0.00	0.00	–	22.74	21.05
		10	23	2.55	3.87	0.15	0.32	0.00	0.00	–	21.43	4.35
		Average		8.07	12.40	2.42	0.42	0.00	0.00	5.84	14.74	7.57
100	25	11	14	209.89	435.85	8.72	1.48	0.00	0.00	0.00	14.57	0.00
		12	13	2.67	28.05	9.67	1.40	0.00	0.00	15.38	11.28	7.69
		13	13	1723.90	151.62	9.13	1.51	0.00	0.00	23.08	19.96	0.00
		14	14	3600.00	35.88	10.12	1.50	15.38	0.00	7.14	7.26	7.14
		15	14	14.01	13.76	10.07	1.54	0.00	0.00	21.43	20.80	14.29
		16	14	40.18	13.41	11.56	1.44	0.00	0.00	7.14	12.72	7.14
		17	14	37.76	59.74	12.55	1.45	0.00	0.00	21.43	26.98	21.43
		18	14	18.74	43.47	10.68	1.42	0.00	0.00	14.29	22.29	0.00
		19	13	17.21	31.33	14.01	1.30	0.00	0.00	30.77	38.37	30.77
		20	*13	3078.14	3600.00	0.58	1.10	23.08	7.69	–	81.80	38.46
		Average		874.25	441.31	9.71	1.41	3.85	0.77	15.63	25.60	12.69
150	37	21	11	363.19	79.51	36.69	2.92	0.00	0.00	27.27	27.07	18.18
		22	11	171.87	103.24	26.91	2.68	0.00	0.00	18.18	34.95	18.18
		23	*11	763.98	3600.01	28.26	2.64	9.09	10.00	27.27	45.86	27.27
		24	11	504.81	166.45	26.09	2.97	0.00	0.00	18.18	17.63	9.09
		25	10	9.10	56.96	15.78	2.64	0.00	0.00	20.00	16.40	10.00
		26	10	12.44	88.53	33.45	2.51	0.00	0.00	30.00	21.40	10.00
		27	12	164.87	111.24	213.91	2.07	0.00	0.00	25.00	68.84	33.33
		28	11	111.83	67.75	31.11	2.51	0.00	0.00	18.18	28.13	9.09
		29	10	6.33	98.90	28.81	2.70	0.00	0.00	20.00	17.60	10.00
		30	10	32.17	116.24	28.23	2.60	0.00	0.00	30.00	23.60	10.00
		Average		214.06	448.88	46.92	2.62	0.91	1.00	23.41	30.15	15.51
200	50	31	9	872.93	194.76	48.09	6.55	0.00	0.00	22.22	23.11	22.22
		32	*9	2152.14	3600.03	9.27	4.64	44.44	11.11	–	287.13	144.44
		33	*9	3473.79	3059.78	71.79	7.16	22.22	12.50	44.44	44.44	44.44
		34	*10	714.62	1489.29	25.50	4.76	20.00	11.11	–	105.89	60.00
		35	10	1645.72	118.15	42.46	6.86	22.22	0.00	20.00	20.28	10.00
		36	9	1282.98	251.35	60.29	7.02	0.00	0.00	33.33	33.76	22.22
		37	9	66.58	183.14	45.71	6.67	0.00	0.00	22.22	19.81	11.11
		38	10	1436.34	507.41	63.14	7.28	11.11	0.00	20.00	25.39	10.00
		39	9	339.13	165.87	46.96	6.39	0.00	0.00	22.22	14.44	11.11
		40	*9	2679.24	1331.90	55.01	7.50	22.22	12.50	33.33	53.56	33.33
		Average		1466.34	1090.17	46.82	6.48	14.22	4.72	27.22	62.78	36.89
		Overall Average		640.68	498.19	26.47	2.73	4.74	1.62	18.03	33.32	18.17

Finally, the results of data set E are displayed in Table 8. We can observe that both the exact methods failed. They no longer solve any instance in 6 h of CPU time. Method ADF provides the best lower bound (f^{lb}) for all instances, but this bound is very poor. In terms of heuristic solution quality, the proposed method obtains better solutions than those obtained by the SPS heuristic using less computational effort.

Table 9 summarizes the comparison among methods. Analyzing this table, we observe that QR is considerably faster than SPS for all data sets. Regarding solution quality, the proposed method provides acceptable solutions for data sets A, B, C and for subsets D-α and D-β. For subsets D-γ and D-δ, the comparison between QR and SPS in terms of solution quality does not make too much sense because SPS fails in finding feasible solutions to a large number of instances. As we can see SPS fails finding feasible solution in 34 instances, whereas QR fails in only 2. Analyzing data set E, QR clearly outperforms all other methods, whose computation time and resources grow considerably. In that regard, our heuristic is still better as it was able to find feasible solutions to practically every instance. When compared to the exact methods over all data sets, our heuristic is still more reliable in terms of number of feasible solutions found. Exact methods ADF and OP failed in finding feasible solutions in 41 and 18 instances, respectively, while QR failed in only 2 instances.

4.4. Component analysis

In this last experiment, we assess the value that each individual component gives to the QR heuristic. We consider the three essential components of the method: Perturbation, VND and Shake. The experiment consists of disabling one component at a time and running the heuristic using 1000 as iteration limit for the data sets A, B. C, and D-*, and 200 as iteration limit for the data set E, with 30 repetitions. The same set of α values of the previous experiment is used. Table 10 displays the comparison of the components for each data set. In this table, the column "All"

Table 6
Comparison of methods on data set D-γ.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR[1]	QR[2]
50	16	01	17	0.12	37.06	1.23	0.55	0.00	0.00	0.00	8.70	0.00
		02	18	15.81	23.62	2.67	0.61	0.00	0.00	5.56	12.13	5.56
		03	15	2.13	18.80	1.99	0.55	0.00	0.00	0.00	47.33	0.00
		04	21	3.92	3.28	3.40	0.64	0.00	0.00	4.76	14.29	14.29
		05	18	0.08	22.87	3.07	0.63	0.00	0.00	0.00	32.96	16.67
		06	19	34.18	2.29	6.69	0.53	0.00	0.00	5.26	32.00	15.79
		07	18	11.46	3.02	7.20	0.57	0.00	0.00	5.56	22.37	5.56
		08	20	10.39	71.76	0.53	0.52	0.00	0.00	–	28.17	20.00
		09	17	5.06	26.33	0.62	0.48	0.00	0.00	–	33.43	11.76
		10	23	6.86	171.56	0.19	0.49	0.00	0.00	–	38.00	13.04
		Average		9.00	38.06	2.76	0.56	0.00	0.00	3.02	26.94	10.27
100	33	11	11	7.85	32.90	12.61	2.47	0.00	0.00	27.27	30.65	9.09
		12	12	37.11	31.91	12.73	2.20	0.00	0.00	8.33	28.50	8.33
		13	12	903.74	59.72	9.68	2.03	0.00	0.00	8.33	19.67	0.00
		14	12	1053.49	48.20	16.82	2.10	0.00	0.00	16.67	28.17	16.67
		15	13	49.23	55.04	11.29	2.14	0.00	0.00	7.69	22.82	7.69
		16	13	1864.90	79.53	17.15	2.20	8.33	0.00	15.38	38.12	15.38
		17	13	408.38	58.80	24.23	1.95	0.00	0.00	7.69	12.69	0.00
		18	14	3600.01	170.91	18.27	2.14	7.69	0.00	7.14	22.38	7.14
		19	*11	3600.00	3600.00	28.15	2.39	9.09	10.00	18.18	67.09	18.18
		20	12	96.22	102.24	4.76	1.67	0.00	0.00	–	95.79	41.67
		Average		1162.09	423.93	15.57	2.13	2.51	1.00	12.96	36.59	12.42
150	50	21	11	3600.00	1592.62	58.20	5.79	20.00	0.00	18.18	49.09	18.18
		22	10	1077.10	159.58	59.59	5.00	0.00	0.00	30.00	120.00	70.00
		23	11	33.99	208.82	32.27	4.33	0.00	0.00	–	114.90	63.64
		24	10	136.11	168.12	44.21	5.84	0.00	0.00	20.00	37.20	30.00
		25	9	98.25	99.75	24.29	6.07	0.00	0.00	11.11	32.66	11.11
		26	9	15.88	107.44	26.44	5.93	0.00	0.00	22.22	54.14	22.22
		27	*11	617.72	3600.00	22.10	4.22	9.09	10.00	–	159.90	54.55
		28	10	3600.00	122.35	28.33	5.70	11.11	0.00	10.00	32.61	10.00
		29	9	6.92	140.94	30.29	5.69	0.00	0.00	22.22	47.22	33.33
		30	9	226.47	194.87	20.30	6.18	0.00	0.00	22.22	48.00	22.22
		Average		941.24	639.45	34.60	5.47	4.02	1.00	19.49	69.57	33.53
200	66	31	8	19.26	95.66	52.80	10.04	0.00	0.00	25.00	40.49	12.50
		32	9	3600.01	1498.20	9.73	6.94	22.22	0.00	–	393.83	244.44
		33	8	3600.00	1531.59	67.81	10.29	12.50	0.00	37.50	64.65	50.00
		34	10	876.98	769.40	67.84	7.17	33.33	0.00	–	185.17	110.00
		35	8	717.64	182.56	65.16	10.52	0.00	0.00	37.50	54.38	37.50
		36	8	3600.01	283.53	40.23	11.42	12.50	0.00	37.50	62.29	37.50
		37	8	21.10	313.41	58.08	10.50	0.00	0.00	25.00	56.32	25.00
		38	*8	3600.00	3600.00	73.45	10.07	12.50	14.29	50.00	70.42	50.00
		39	8	27.09	259.93	46.90	10.06	0.00	0.00	25.00	27.92	25.00
		40	9	489.26	302.57	70.91	11.02	12.50	0.00	22.22	54.26	33.33
		Average		1655.14	883.69	55.29	9.80	10.56	1.43	32.47	100.97	62.53
		Overall Average		941.87	496.28	27.05	4.49	4.27	0.86	16.99	58.52	29.68

Table 7
Comparison of methods on data set D-δ.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR^1	QR^2
50	20	01	17	0.23	107.37	1.78	0.64	0.00	0.00	0.00	62.23	5.88
		02	18	0.70	78.12	4.24	0.95	0.00	0.00	5.56	59.93	16.67
		03	17	2.38	87.34	6.29	0.80	0.00	0.00	0.00	130.85	41.18
		04	20	29.45	102.04	4.71	0.74	0.00	0.00	10.00	44.00	20.00
		05	19	21.26	63.37	0.36	0.59	0.00	0.00	–	88.76	42.11
		06	21	4.12	18.01	0.22	0.60	0.00	0.00	–	60.93	40.00
		07	19	4.81	17.83	0.23	0.65	0.00	0.00	–	70.10	10.53
		08	23	3600.00	159.20	0.23	0.59	20.00	0.00	–	39.48	4.35
		09	21	21.21	68.26	0.22	0.58	0.00	0.00	–	92.28	42.86
		10	23	14.20	52.95	0.24	0.60	0.00	0.00	–	159.78	82.61
		Average		369.84	75.45	1.85	0.67	2.00	0.00	3.89	80.83	30.62
100	40	11	12	52.88	125.81	37.62	2.77	0.00	0.00	16.67	76.48	41.67
		12	11	3600.01	89.80	25.44	3.18	20.00	0.00	18.18	74.30	36.36
		13	12	284.85	138.65	22.88	2.71	0.00	0.00	0.00	73.80	41.67
		14	12	12.69	90.96	70.71	2.21	0.00	0.00	–	121.62	58.33
		15	13	4.25	145.28	5.78	2.35	0.00	0.00	–	130.92	69.23
		16	13	272.28	143.99	103.63	2.23	0.00	0.00	–	66.41	30.77
		17	14	3600.01	154.08	5.49	2.10	7.69	0.00	–	134.98	64.29
		18	14	3600.01	209.06	3.39	2.02	7.69	0.00	–	111.15	50.00
		19	12	3600.00	289.10	29.40	2.15	9.09	0.00	–	109.83	58.33
		20	14	73.41	509.37	3.93	2.07	0.00	0.00	–	278.57	164.29
		Average		1510.04	189.61	30.83	2.38	4.45	0.00	11.62	117.81	61.49
150	60	21	11	3600.00	290.33	242.81	8.12	20.00	0.00	18.18	119.80	63.64
		22	12	3600.00	2930.40	10.23	5.20	9.09	0.00	–	275.37	116.67
		23	11	270.09	231.48	5.87	5.20	0.00	0.00	–	273.67	127.27
		24	9	3002.52	180.47	81.27	6.88	0.00	0.00	33.33	120.43	66.67
		25	9	5.21	142.60	24.77	7.85	0.00	0.00	11.11	73.89	22.22
		26	9	43.51	194.92	36.84	10.51	0.00	0.00	22.22	86.36	44.44
		27	*13	3600.04	3600.00	7.74	5.08	7.69	11.11	–	238.46	238.46
		28	10	3600.00	1615.84	35.98	7.47	11.11	0.00	20.00	72.17	30.00
		29	9	8.04	232.82	38.47	8.56	0.00	0.00	22.22	91.05	44.44
		30	9	14.84	240.06	31.32	7.58	0.00	0.00	11.11	77.22	33.33
		Average		1774.43	965.89	51.53	7.24	4.79	1.11	19.74	142.84	78.71
200	80	31	8	26.43	233.07	55.32	13.82	0.00	0.00	37.50	75.00	37.50
		32	*10	3600.03	1277.43	11.98	8.45	40.00	37.50	–	–	–
		33	8	84.32	402.58	107.37	12.38	0.00	0.00	37.50	112.57	62.50
		34	12	1577.88	421.93	11.98	8.73	40.00	0.00	–	–	–
		35	*8	3600.00	1318.10	125.24	11.40	12.50	14.29	50.00	124.79	62.50
		36	9	57.93	171.37	403.80	11.61	0.00	0.00	–	135.37	66.67
		37	8	32.87	334.25	72.87	14.14	0.00	0.00	25.00	109.10	62.50
		38	8	3600.00	433.88	18.90	9.34	12.50	0.00	–	245.88	100.00
		39	8	85.02	302.22	42.66	17.40	0.00	0.00	25.00	80.90	37.50
		40	*8	3600.01	927.86	94.24	11.64	12.50	14.29	–	186.80	87.50
		Average		1626.45	582.27	94.44	11.89	11.75	6.61	35.00	133.80	64.58
		Overall Average		1320.19	453.30	44.66	5.55	5.75	1.93	17.56	118.82	58.85

Table 8
Comparison of methods on data set E.

n	p	Instance	Optimal	Time (s)				Deviation (%)				
				ADF	OP	SPS	QR	ADF	OP	SPS	QR^1	QR^2
3038	600	R1	*77	21 600.00	1167.03	1038.46	192.35	93.51	–	532.47	115.54	88.31
3038	700	R2	*58	21 600.00	1167.04	941.10	246.38	156.90	–	406.90	177.59	165.52
3038	800	R3	*58	21 600.00	1167.05	2975.48	326.45	156.90	–	372.41	201.03	163.79
3038	900	R4	*58	21 600.00	1167.10	857.26	462.34	156.90	–	432.76	240.75	182.76
3038	1000	R5	*40	21 600.00	1167.05	3575.76	524.59	272.50	–	485.00	471.25	352.50
		Overall average		21 600.00	1167.06	1877.61	350.42	167.34	–	445.91	241.23	190.58

represents the average GAP when all components are enabled, which matches the value displayed in Table 9, column QR^2. Each column in the section "Components" represent the component disabled during the experiment and shows the average GAP value obtained. The section "Contribution %" displays the percentage value that the specific component provides with respect to total value shown in "All". For instance, in the row associated to data set A, QR obtains solutions that have an average GAP of 0.23%. When QR is run with the Perturbation component disabled, the average deviation worsens to 17.25%. This represents a 77.25% contribution

as indicated in the Contribution (%), computed as follows:

$$\text{Contribution} = \frac{\text{Perturbation} - \text{All}}{\sum_{i=1}^{c}(\text{GAP}_i - \text{All})} \times 100, \tag{11}$$

where c is the total number of components, and GAP_i is the average quality solution obtained when the i-th component is disabled. It is remarkable that the most influential component within the heuristic is the Perturbation, followed by VND and Shake. This is consistent with the statistical analysis, which showed that the parameter α, used in the perturbation, influences the response variable. Nevertheless, all other two components add value to the overall performance. The benefit of VND ranges from 8.32% to 16.53%, and the benefit of the Shake method ranges from 0.64% to 24.12%.

4.5. Asymptotic analysis

Fig. 2 shows a comparison of the methods in terms of their asymptotic running time and used memory resources with respect to the instance size. The memory statistic indicates the maximum resident set size used [20], in bits, that is, the maximum number of bits of physical memory that each approach used simultaneously. As can be seen, the resources used by the proposed approach are lower than those used by the other three methods. In particular, the memory usage requirements of the two exact methods, as expected, are considerably larger.

5. Conclusions

We have proposed a metaheuristic framework that integrates several components such as a greedy randomized procedure with adaptive probabilistic selection in its construction phase and iterated greedy local search with a variable neighborhood descent in its local search phase. The results indicate that the proposed heuristic outperforms the best heuristic in terms of solution quality, running time, and reliability on finding feasible solutions in harder instances. The performance of the proposed approach is more robust than that of the exact methods, requiring less computational effort and memory for obtaining solutions of reasonably good quality for data sets A, B, and C. For the harder instances in data sets D-$\gamma$and D-δ, the optimality gaps of the

Table 10
Component-wise analysis for heuristic QR.

Data set	All	Components			Contribution (%)		
		Perturbation	VND	Shake	Perturbation	VND	Shake
A	0.23	17.25	2.77	2.70	77.25	11.52	11.23
B	3.41	12.53	4.66	5.33	74.19	10.16	15.65
C	3.58	36.63	8.32	15.59	66.38	9.51	24.12
D-α	2.72	31.87	5.61	5.39	83.98	8.32	7.70
D-β	18.17	106.97	33.19	29.04	77.42	13.09	9.48
D-γ	29.68	195.23	46.81	45.03	83.60	8.65	7.75
D-δ	58.85	261.94	105.76	92.62	71.57	16.53	11.90
E	190.58	496.62	245.65	192.91	84.20	15.15	0.64

Table 9
Summary of comparison among methods on all data sets.

Data set	Average deviation (%)				Average time (s)				Number of failures			
	ADF	OP	SPS	QR[2]	ADF	OP	SPS	QR	ADF	OP	SPS	QR
A	0.00	0.00	5.41	0.23	27.53	33.46	5.34	0.39	0	0	0	0
B	0.00	0.00	6.70	3.41	178.53	154.52	27.30	0.76	0	0	0	0
C	0.00	0.00	31.79	3.58	233.63	414.76	285.43	2.39	0	0	0	0
D-α	0.18	0.00	12.62	2.72	255.36	139.72	25.22	0.84	1	0	0	0
D-β	4.74	1.62	18.03	18.17	640.68	498.19	26.47	2.73	9	6	5	0
D-γ	4.27	0.86	16.99	29.68	941.87	496.28	27.05	4.49	12	3	8	0
D-δ	5.75	1.93	17.56	58.85	1320.19	453.30	44.66	5.55	14	4	21	2
E	167.34	–	445.91	190.58	21 600.00	1167.06	1877.61	350.42	5	5	0	0

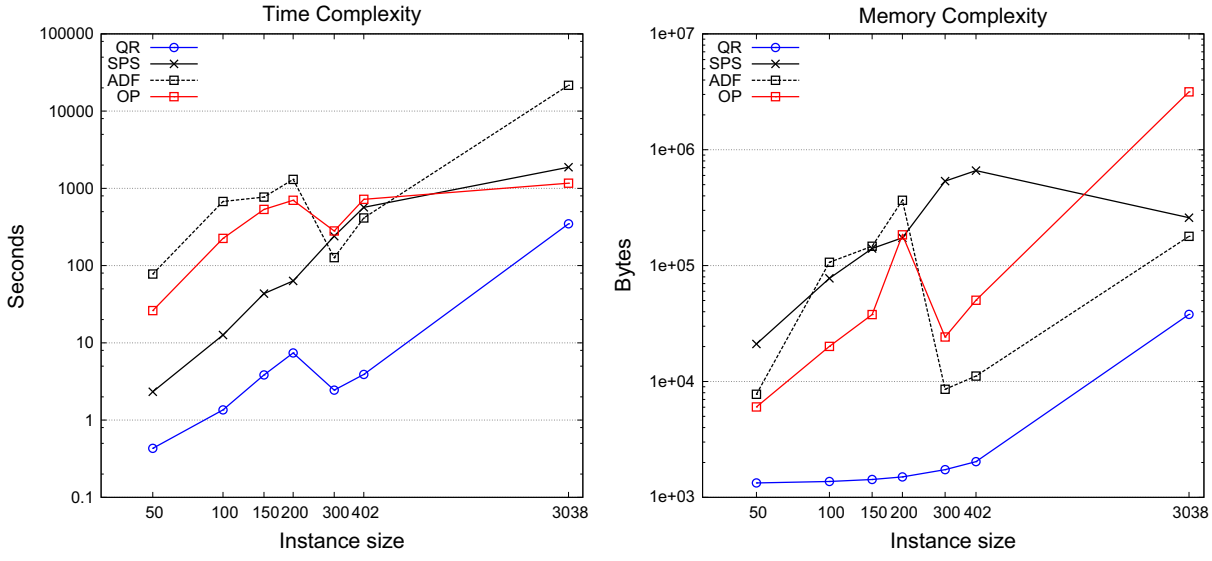

Fig. 2. Comparison among methods in terms of asymptotic running time and memory usage. The axes are on a logarithmic scale.

heuristic solutions are not as good; however, they are still obtained very quickly. For this harder set, our heuristic found feasible solutions to almost all instances tested, which is clearly superior to the SPS heuristic as it failed in several instances. For the large scale data set E, both exact methods failed in finding optimal solutions to all instances tested. Our heuristic found better solutions than the best integer solution reported by the exact methods. Our heuristic clearly outperformed SPS as well in this data set. In a detailed component analysis, we have seen that the success of the heuristic is mainly due to the perturbation and VND methods. However, for data sets A, B, and C, the shake method proved very worthwhile as well.

The proposed method provides robust solutions in a short time to the problems previously discussed in the literature. For data sets D and E analyzed in this paper, the method ensures greater success in finding feasible solutions than that of the existing heuristic.

Acknowledgments

We are very grateful to the anonymous reviewers for their criticism which helped improve the quality of this work. This work was supported by the Mexican National Council for Science and Technology (Grant CONACYT CB-2011-01-166397) and Universidad Autónoma de Nuevo León (Grant UANL-PAICYT CE728-11). We also thank Maria Scaparra and Juan A. Díaz for providing us with the source code of their methods.

References

[1] Kariv O, Hakimi SL. An algorithmic approach to network location problems, I: the *p*-centers. SIAM J Appl Math 1979;37(3):513–38.
[2] Elloumi S, Labbé M, Pochet Y. A new formulation and resolution method for the *p*-center problem. INFORMS J Comput 2004;16(1):84–94.
[3] Özsoy FA, Pınar MÇ. An exact algorithm for the capacitated vertex *p*-center problem. Comput Oper Res 2006;33(5):1420–36.
[4] Albareda-Sambola M, Díaz JA, Fernández E. Lagrangean duals and exact solution to the capacitated *p*-center problem. Eur J Oper Res 2010;201(1):71–81.
[5] Scaparra MP, Pallottino S, Scutellà MG. Large-scale local search heuristics for the capacitated vertex *p*-center problem. Networks 2004;43(4):241–55.
[6] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur J Oper Res 2007;177(3):2033–49.
[7] Lourenço HR, Martin O, Stützle T. Iterated local search. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Norwell: Kluwer Academic Publishers; 2002. p. 321–53.
[8] Ruiz R, Stützle T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. Eur J Oper Res 2008;187(3):1143–59.
[9] Fanjul-Peyro L, Ruiz R. Iterated greedy local search methods for unrelated parallel machine scheduling. Eur J Oper Res 2010;207(1):55–69.
[10] Urlings T, Ruiz R, Stützle T. Shifting representation search for hybrid flexible flowline problems. Eur J Oper Res 2010;207(2):1086–95.
[11] Hansen P, Mladenović N. An introduction to variable neighborhood search. In: Voss S, Martello S, Osman IH, Roucairol C, editors. Meta-heuristics: advances and trends in local search paradigms for optimization. Boston, USA: Kluwer; 1999. p. 433–58.
[12] Subramanian A, Drummond LMA, Bentes C, Ochi LS, Farias R. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. Comput Oper Res 2010;37(11):1899–911.
[13] Martins AX, Duhamel C, Souza MC, Saldanha RR, Mahey P. A VND-ILS heuristic to solve the RWA problem. In: Pahl J, Reiners T, Voß S, editors. Network optimization. Lecture notes in computer science, vol. 6701. Berlin, Germany: Springer; 2011. p. 577–82.
[14] Quevedo-Orozco DR, Ríos-Mercado RZ. A new heuristic for the capacitated vertex *p*-center problem. In: Bielza C, Salemrón A, Alonso-Betanzos A, Hidalgo JI, Martínez L, Troncoso A, et al., editors. Advances in artificial intelligence. Lecture notes in artificial intelligence, vol. 8109. Heidelberg, Germany: Springer; 2013. p. 279–88.
[15] Beasley JE. OR-Library: distributing test problems by electronic mail. J Oper Res Soc 1990;41(11):1069–72.
[16] ReVelle CS, Eiselt HA. Location analysis: a synthesis and survey. Eur J Oper Res 2005;165(1):1–19.
[17] Lorena LAN, Senne ELF. A column generation approach to capacitated *p*-median problems. Comput Oper Res 2004;31(6):863–76.
[18] Ceselli A, Righini G. A branch-and-price algorithm for the capacitated *p*-median problem. Networks 2005;45(3):125–42.
[19] Reinelt G. The traveling salesman: computational solutions for TSP applications. Lecture notes in computer science, vol. 840. Heidelberg: Springer; 1994.
[20] Loosemore S, Stallman RM, McGrath R, Oram A, Drepper U. The GNU C library reference manual: for version 2.17. Free Software Foundation; 2012.