A New Heuristic for the Capacitated Vertex p-Center Problem

Dagoberto R. Quevedo-Orozco and Roger Z. Ríos-Mercado

Universidad Autónoma de Nuevo León, Graduate Program in Systems Engineering, Cd. Universitaria, San Nicolás de los Garza, NL 66450, México
{dago,roger}@yalma.fime.uanl.mx

Abstract. A metaheuristic for the capacitated vertex p-center problem is presented. This is a well-known location problem that consists of placing p facilities and assigning customers to these in such a way that the largest distance between any customer and its associated facility is minimized. In addition, a capacity on demand for each facility is considered. The proposed metaheuristic framework integrates several components such as a greedy randomized adaptive procedure with biased sampling in its construction phase and iterated greedy with a variable neighborhood descent in its local search phase. The overall performance of the heuristic is numerically assessed on widely used benchmarks on location literature. The results indicate the proposed heuristic outperforms the best existing heuristic.

Keywords: Combinatorial optimization, discrete location, metaheuristics, GRASP, IGLS, VND.

1 Introduction

The vertex p-center problem can be defined as the problem of locating p facilities and assigning customers to them so as to minimize the longest distance between any customer and its assigned facility. The term *vertex* means that the set of candidate facility sites and the set of customers are the same. In the capacitated version (CpCP) it is required that the total customer demand assigned to each facility does not exceeded its given capacity. The CpCP is $\mathcal{NP}$-hard [1]. Practical applications of p-center problems can be found in school districting planning or system design in health coverage, to name a few.

The uncapacitated version of the problem has been widely investigated from both exact and approximate approaches. Elloumi et al. [2] provide an extensive review of the literature. The CpCP has received less attention in the literature. From an exact optimization perspective, Özsoy and Pınar [3] presented an exact method based on solving a series of set covering problems using an off-the-shelf mixed-integer programming (MIP) solver while carrying out an iterative search over the coverage distances. More recently, Albareda-Sambola et al. [4] proposed an exact method based on Lagrangian relaxation and a covering reformulation. From the heuristic perspective, the work of Scaparra et al. [5] stands as the

C. Bielza et al. (Eds.): CAEPIA 2013, LNAI 8109, pp. 279–288, 2013.

most significant. They developed a heuristic based on large-scale local search with a multiexchange neighborhood represented by an improved graph exploiting principles from network optimization theory. In this paper, we present a metaheuristic framework that integrates several components such as greedy randomized adaptive procedures with biased sampling in its construction phase and iterated greedy with a variable neighborhood descent in its local search phase. The empirical work indicates our heuristic outperforms the best existing method.

2 Problem Formulation

Let V be the set of nodes representing customers or potential locations for the p facilities. The integer distance between nodes i and j is represented for d_{ij}. Each node $j \in V$ has a demand or weight w_j and each node $i \in V$ has a capacity defined by s_i. For the combinatorial model, a p-partition of V is denoted by $X = \{X_1, ..., X_p\}$, where $X_k \subset V$ is called 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 $X_k \cap X_l = \varnothing$ for all $k, l \in K, k \neq l$ where $K = \{1, ..., p\}$. The set of centers is denoted by $P \subset V$ such that $P = \{c(1), ..., c(p)\}$ where $c(k)$ is the active location for subset X_k, i.e., the node that hosts the facility serving the customers in X_k. The problem can be represented by the following combinatorial model.

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

where Π is the collection of all p-partitions of V. For a given territory X_k its cost function, also called the bottleneck cost, is computed as $f(X_k) = \max_{j \in X_k} \{d_{j,c(k)}\}$ where the center $c(k)$, taking into account the capacity, is given by

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

Here, by convention, if for a given X_k there is not any $i \in X_k$ such that $\sum_{j \in X_k} w_j \leq s_i$ then $f(X_k) = \infty$.

3 Proposed Heuristic

To solve the problem we propose a metaheuristic framework with several components such as a greedy randomized adaptive [6] procedure with biased sampling in its construction phase and Iterated Greedy Local Search (IGLS) with a Variable Neighborhood Descent (VND) in its local search phase. IGLS is a method related to the Iterated Local Search (ILS) originally proposed by Ruiz and Stützle [7]. IGLS takes a solution as an input and iteratively applies destruction and reconstruction phase, in a special way focusing on the space of solutions that are locally optimal. Instead of iterating over a local search as done in ILS, IGLS iterates over a greedy reconstruction heuristic.

The VND is a variant of Variable Neighborhood Search (VNS) proposed by Hansen and Mladenovic [8, 9]. VNS is a metaheuristic for solving combinatorial and global optimization problems whose basic idea is a systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. VND method is obtained if a change of neighborhoods is performed in a deterministic way. The proposed approach is presented in Algorithm 1. An initial solution is obtained on Steps 2–3. Within the main loop (Steps 5–14), the local search (Steps 6–7) is performed as long as the solution keeps improving. By improving we mean that either the new solution has a better objective function than the previous or if it reduces the number of bottleneck customers while not worsening the total cost, without creating new bottleneck subsets and new bottleneck customers. If the solutions does not improve, then a shake of the solution is applied, this is defined as removing several bottleneck subsets that meet a given criteria and reconstructing a new solution from the partial solution. These components are described next.

Algorithm 1. GVND

```
 1: procedure GVND(V, p, α, β, Iter_max, LB)
 2:     X ← Construction(α, p)
 3:     X ← VND(X)
 4:     X^best ← X
 5:     while ¬(stopping criteria) do
 6:         X ← IGLS(β, X)
 7:         X ← VND(X)
 8:         if X is better that X^best then
 9:             X^best ← X
10:         else
11:             X ← Shake(α, X)
12:         end if
13:         Iter_max ← Iter_max − 1
14:     end while
15:     return X^best
16: end procedure
```

Construction: The construction phase is comprised of two sub-tasks: (a) center location and (b) customer allocation. First, p nodes are chosen as centers. The choice of these centers is made through a greedy randomized adaptive construction procedure, taking into account the distance factors and the capacity of each vertex $j \in V$. This phase is based on the greedy method proposed by Dyer [10] for the p-center problem. The location phase starts by choosing the first center randomly. Then, we iteratively choose the next center seeking a node whose weighted distance from its nearest center is relatively large. The motivation of this is to try to obtain centers that are as disperse as possible, but also to favor the choice of centers with large capacity such we can assign more customers to

it in the allocation phase. Within a greedy randomized procedure method this is done as follows. Let P be a partial set of chosen centers. Then for each $j \in V \setminus P$, its nearest center is given by $i^* = \arg\min_{i \in P}\{d_{ij}\}$. The we compute the greedy function as

$$\gamma(j) = s_j d_{i^* j} \tag{3}$$

A restricted candidate list (RCL) is built by the elements whose greedy function evaluation falls, within $\alpha\%$ of the best value. RCL $= \{j : \gamma(j) \geq \gamma^{\max} - \alpha(\gamma^{\max} - \gamma^{\min})\}$, where $\alpha \in (0, 1)$.

Instead of choosing the next candidate element to add to the partial solution uniformly at random, we introduce a biased selection mechanism. In the construction mechanism proposed by Bresina [11], a family of such probability distributions is introduced. First, a rank $r[j]$ assigned to each candidate element j, according to its greedy function value (3). The element with the largest greedy function value has rank 1, the second largest has rank 2, and so on. In this case, we defined the bias function using an exponential distribution as $b(r[j]) = e^{-r[j]}$. Once all elements of the RCL have been ranked, the probability $\pi(j)$ of selecting element $j \in$ RCL can be computed as $\pi(j) = b(r[j]) / \sum_{j' \in \text{RCL}} b(r[j'])$.

Once the centers are fixed, the second sub-task consists of allocating the customers to these centers. This phase is performed in a deterministic greedy manner. As some preliminary testing showed, performing this step under a randomized greedy strategy did not bring any value to the quality of the solution. In addition, the pure greedy approach in this phase is more efficient. The customers 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 follows:

$$\phi(j, k) = \max\left\{\frac{d_{jc(k)}}{\bar{d}}, -\left(s_{c(k)} - \sum_{j' \in X_k} w_{j'}\right) + w_j\right\} \tag{4}$$

where $\bar{d} = \max_{i,j \in V}\{d_{ij}\} + 1$ is a normalization factor. If the capacity constraint is satisfied, the function only takes into account the distance factor, otherwise, the function returns an integer value that penalizes the assignment. Then assigns each node j to a nearest center, namely $X_{k^*} \leftarrow X_{k^*} \cup \{j\}$ where $k^* = \arg\min_{k \in K} \phi(j, k)$. Finally, once the assignment is done, the centers for the entire partition are updated using (2).

Local Search: Given an initial solution built by the construction phase, the improvement phase applies an IGLS followed by VND with two neighborhoods based on insertion and exchange. Each procedure is briefly described next.

1. *IGLS:* This method takes a solution as an input and iteratively applies destruction and reconstruction phases. In this specific case, deallocating the $\beta\%$ of nodes located in X_k, with high values of the function $\rho(j) = d_{jc(k)} / \sum_{j' \in X_k} d_{jc(k)}$. 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. The reconstruction phase reassigns each disconnected node

to a nearest center, namely $X_{k^*} \leftarrow X_{k^*} \cup \{j\}$ where $k^* = \arg\min_{k \in K} \phi(j,k)$. A priority assignment is given to the bottleneck nodes, i.e., nodes whose previous assignment matched the value of the objective function value.

2. *VND:* This method is formed by two neighborhoods based on reinsertion and exchange movements. It is presented in Algorithm 2, where neighborhoods are denoted as $\mathcal{N}_k, k = 1, ..., k_{\max}$, in this case $k_{\max} = 2$. For each of the two neighborhoods, the potential move takes into account the distance factors and the capacity. Each neighborhood is briefly described next.

Algorithm 2. Variable Neighborhood Descent

```
1: procedure VND(X)
2:     while k ≤ k_max do
3:         X' ← arg min_{y∈N_k(X)} f(y)
4:         if X' is better that X then
5:             X' ← X
6:             k ← 1
7:         else
8:             k ← k + 1
9:         end if
10:    end while
11:    return X
12: end procedure
```

$\mathcal{N}_1$) *Reinsertion:* This neighborhood considers moves where a node i (currently assigned to center of set X_q) is assigned to set X_k, i.e., given $X = (X_1, \ldots, X_p)$ $reinsertion(i,k) = \{X_1, \ldots, X_q \setminus \{i\}, \ldots, X_k \cup \{i\}, ..., X_p\}$ where i must be a bottleneck node for the move to be attractive.

$\mathcal{N}_2$) *Exchange:* This neighborhood considers moves where two nodes i and j in different subsets are swaped, i.e., given $X = (X_1, \ldots, X_p)$, $swap(i,j) = \{X_1, ..., X_q \cup \{j\} \setminus \{i\}, ..., X_k \cup \{i\} \setminus \{j\}, ..., X_p\}$, where either i or j must be a bottleneck node for the move to be attractive.

Improvement Criteria: We uses a effective improvement criteria propose in [5] which includes the reduction of bottleneck elements, this is defined as

$$f(X') < f(X) \vee (f(X') = f(X), \mathcal{B}(X') \subseteq \mathcal{B}(X), \mathcal{J}(X') \subset \mathcal{J}(X)) \qquad (5)$$

where $\mathcal{B}(X)$ denote 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 location in each subset X_k, i.e., $\mathcal{J}(X) = \{j \in X_k : d_{jc(k)} = f(X), k \in \mathcal{B}(X)\}$. This criteria is met if it decreases the objective function value or if it 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^{best} is updated if a better feasible solution is found according to the criterion (5) otherwise a shake of the solution X is applied.

Shake: We define an auxiliary mechanism that performs a partial shake of the current solution through an aggressive removal and reconstruction of several subsets, which diversifies the structure of the solution. The selection criteria of subsets is

$$\mathcal{L} \leftarrow \{\eta^1(j), \eta^2(j), \eta^3(j) : \eta^1(j) = l(j), j \in \mathcal{J}(X)\} \tag{6}$$

where $\eta(j) = \arg\min_{k \in K} d_{jc(k)}$. Then $\eta^1(j)$, $\eta^2(j)$, and $\eta^3(j)$ are the first, second, and third nearest centers to j, respectively, under the distance criterion. $l(j)$ is the center serving customer j. Let $W \leftarrow \cup_{k \in \mathcal{L}} X_k$. We then now remove these sets from the current solution $X \leftarrow X \setminus W$. Now, using the construction phase, we construct a new solution X' by reassigning the nodes in W with $p = |\mathcal{L}|$. Finally $X \leftarrow X \cup X'$ is the new current solution.

Stopping criteria: The approach stops when the maximum number of iterations is met or if a relative deviation with respect to a known (if any) lower bound (LB) for the problem is less than a given ϵ. For our practical purposes a value of 1.0×10^{-8} is used for ϵ.

4 Computational Results

This section shows the overall performance of the heuristic which is empirically assessed on widely used benchmarks on location literature. The heuristic was coded in C++, compiled with gcc/g++ version 4.2 with the "-O3" optimization level. ILOG CPLEX 12.5 is used in exact method proposed in [3] and we imposed some resource limitation to every test: computation was halted after 1 hour or in case of memory overflow. Each of the experiments was carried out on a MacBook Pro 13" with Intel Core i5 2.4 GHz, 4 GiB RAM under OS X Lion 10.7.5. For the experiments, we used three different data sets generated for other location problems.

(Set A) Beasley OR-Library: Contains two groups of 10 instances, with 50 demand nodes and 5 facilities to be located, and 100 demand nodes and 10 facilities to be located, respectively. In all of the problems the capacity is assumed equal for every facility.

(Set B) Galvão and ReVelle: The set includes two networks that were randomly generated by for the maximal covering location problem. The set includes 8 instances with size of 100 and 150 customers, and range from 5 to 15 centers. In this case, the facility capacities are variable.

(Set C) Lorena and Senne: The set includes 6 large instances whose size ranges from 100 to 402 customers, and from 10 to 40 centers. Also in this case, all of the facility sites have equal capacity. This set is considered large scale and therefore more difficult to solve.

Recall from Section 3 that two important algorithmic parameters are α and β. In a preliminary phase, the heuristic was fine-tuned by running the algorithm 50 iterations for each possible combination $\alpha \times \beta \in \{0.0, 0.1, \ldots, 1.0\} \times \{0.0, 0.1, \ldots, 1.0\}$

on the three data sets. We choose the best combination (α, β) for each dataset based on an average of the objective function value for all executions and combinations (α, β). For the remaining experiments, the choices of (α, β) are set to (0.4, 0.3), (0.3, 0.3), and (0.3,0.3), for data sets A, B, and C, respectively.

For the next experiments every run of our heuristic was done using 30 repetitions with different random seeds and using 500 as iteration limit. No lower bound (LB) was used. We perform a comparison of the proposed approach (Heuristic QR) with the heuristic by Scaparra et al. [5] (Heuristic SP) and the exact method by Özsoy and Pınar [3] (Exact OP). These methods have been executed over the same machine, under the conditions specified for each method, to ensure a fair comparison.

Table 1. Comparison of methods on data set A

n	p	Instance	Optimal	OP		SP		QR		
				gap %	Time (s)	gap %	Time (s)	gap^1 %	gap^2 %	Time (s)
50	5	cpmp01	29	0.00	0.19	0.00	0.45	0.00	0.00	0.45
		cpmp02	33	0.00	1.13	0.00	0.72	0.00	0.00	0.49
		cpmp03	26	0.00	0.20	0.00	0.56	0.00	0.00	0.48
		cpmp04	32	0.00	0.53	0.00	0.61	0.00	0.00	0.53
		cpmp05	29	0.00	1.02	0.00	0.69	0.00	0.00	0.50
		cpmp06	31	0.00	1.62	3.23	0.75	2.90	0.00	0.49
		cpmp07	30	0.00	0.51	0.00	0.91	0.67	0.00	0.53
		cpmp08	31	0.00	0.61	0.00	0.73	0.00	0.00	0.49
		cpmp09	28	0.00	0.74	3.57	0.91	3.33	0.00	0.49
		cpmp10	32	0.00	2.14	12.50	1.74	13.75	0.00	0.48
		Average		0.00	0.87	1.93	0.81	2.07	0.00	0.49
100	10	cpmp11	19	0.00	2.91	21.05	5.4	8.25	0.00	1.41
		cpmp12	20	0.00	2.91	10.00	5.74	4.17	0.00	1.39
		cpmp13	20	0.00	3.46	5.00	5.46	0.33	0.00	1.36
		cpmp14	20	0.00	2.15	10.00	5.28	2.50	0.00	1.37
		cpmp15	21	0.00	4.06	9.52	5.9	3.49	0.00	1.41
		cpmp16	20	0.00	6.96	10.00	7.04	3.83	0.00	1.43
		cpmp17	22	0.00	30.14	9.09	6.03	4.55	4.55	1.41
		cpmp18	21	0.00	6.50	4.76	4.74	1.75	0.00	1.34
		cpmp19	21	0.00	9.30	9.52	6.25	5.40	0.00	1.42
		cpmp20	21	0.00	12.25	0.00	5.93	8.89	0.00	1.44
		Average		0.00	8.06	8.90	5.78	4.31	0.45	1.40
		Overall average		0.00	4.47	5.41	3.29	3.19	0.23	0.94

Tables 1–3 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. For each method column "gap (%)" expresses the percent of relative deviation or gap with respect to the optimal value and "Time (s)" gives the execution time in seconds. It should be noted that for the proposed method QR, we show the time average performance over the 30 independent repetitions, also "gap^1 %" and "gap^2 %" denote the average and best gap, respectively, over all repetitions. Table 4 summarizes the comparison among methods for the three data sets in terms of their average relative optimality gap, running time, and memory usage. The memory statistic indicates the maximum resident set size used [12], in bits, that is, the maximum number of bits of physical memory that each approach used simultaneously.

Table 2. Comparison of methods on data set B

n	p	Instance	Optimal	OP		SP		QR		
				gap %	Time (s)	gap %	Time (s)	gap^1 %	gap^2 %	Time (s)
100	5	G1	94	0.00	4.49	3.19	4.71	1.88	1.06	1.39
100	5	G2	94	0.00	5.90	3.19	4.48	1.60	0.00	1.23
100	10	G3	83	0.00	121.44	9.64	8.01	8.72	4.82	1.58
100	10	G4	84	0.00	25.03	8.33	8.28	8.73	5.95	1.54
150	10	G5	95	0.00	190.95	5.26	22.61	4.95	3.16	2.70
150	10	G6	96	0.00	120.46	5.21	21.21	4.38	3.13	2.37
150	15	G7	89	0.00	60.62	8.99	28.31	8.35	5.62	3.53
150	15	G8	89	0.00	213.61	10.11	26.52	8.84	6.74	3.48
		Overall average		0.00	92.81	6.74	15.52	5.93	3.81	2.23

The first thing to notice is that for all instances tested, an optimal solution was found by the exact method, such that the "gap" column in all tables represents the true relative optimality gap found by any method. As far as data set A is concerned, the exact method was 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 1.5 seconds to solve each instance. In terms of solution quality, the proposed heuristic found better solutions than the ones reported by the SP heuristic.

Table 3. Comparison of methods on data set C

n	p	Instance	Optimal	OP		SP		QR		
				gap %	Time (s)	gap %	Time (s)	gap^1 %	gap^2 %	Time (s)
100	10	SJC1	364	0.00	195.16	26.67	8.79	23.24	7.478	0.68
200	15	SJC2	304	0.00	74.30	10.48	39.60	7.37	1.599	1.95
300	25	SJC3a	278	0.00	136.49	38.73	125.03	16.41	7.184	6.12
300	30	SJC3b	253	0.00	152.20	35.59	119.65	13.16	3.661	8.38
402	30	SJC4a	284	0.00	522.63	30.99	283.18	9.76	5.219	11.39
402	40	SJC4b	239	0.00	157.52	44.12	241.68	10.94	2.346	18.56
		Overall average		0.00	206.38	31.10	136.32	13.48	4.58	7.85

When analyzing data set B we can observe that the exact method takes considerably longer than both heuristics to reach an optimal solution. On average, the exact method takes about an order of magnitude longer. In terms of solution quality, again our heuristic obtains better solutions (average gap of 5.93 %) than the SP heuristic (average gap of 6.74%). Regarding data set C, we can observe that the exact method takes on average above 4 minutes while our heuristic takes less than 9 seconds. When comparing our heuristic with the SP heuristic, we can see that ours is faster and finds solutions of significantly better quality. Figure 1 shows a comparison of the methods in terms of their asymptotic running time and used memory resources with respect to the number of nodes. As can be seen, the resources used by the proposed approach are lower than those used by the other two methods.

There exist a recent data set added to the OR-Library that features values of p proportional to the number of nodes. This is regarded as a very hard set to

Table 4. Summary of comparison among methods on data sets A, B, and C

Dataset	Average gap (%)				Average time (s)			Average memory (bits)		
	OP	SP	QR[1]	QR[2]	OP	SP	QR	OP	SP	QR
A	0.00	5.41	3.19	0.23	4.47	3.29	0.94	2.59E+07	4.37E+07	5.12E+05
B	0.00	6.74	5.93	3.81	92.81	15.52	2.23	4.78E+07	2.12E+08	5.57E+05
C	0.00	31.10	13.48	4.58	206.38	136.32	7.85	1.38E+08	4.70E+08	7.96E+05

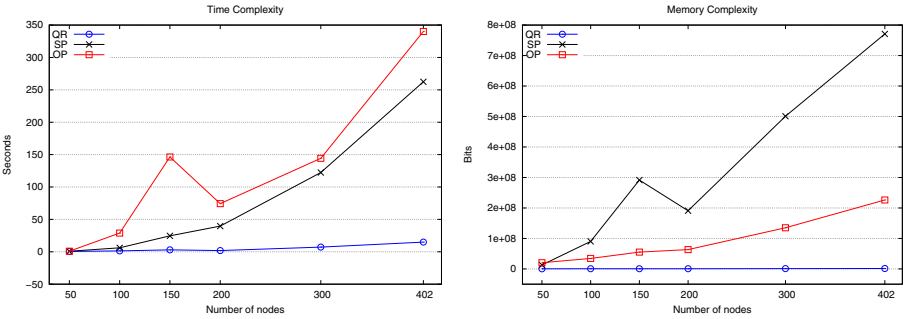

Fig. 1. Comparison of methods in terms of asymptotic running time and memory usage

solve for capacitated location problems such as p-median and p-center problems. In our preliminary experiments, we have observed that the exact method fails to find optimal solutions for some instances. Table 5 displays the results on this data set. As can be seen, the exact method is unable to find a feasible solution in all five instances either by reaching the time limit of 1 hr (instances 1 and 3) or by running out of memory (instances 2, 4, and 5). Heuristic SP fails in delivering an optimal solution in 3 out of 5 instances. Our heuristic finds a feasible solution in 4 out of 5 instances.

Table 5. Comparison of methods on data set D

Subset	Instance	n	p	OP		SP		QR	
				Best LB	Time (s)	Objetive	Time (s)	Objetive	Time (s)
D	27	150	60	10	3600.00	-	-	55	22.94
D	32	200	80	11	959.04	-	-	-	-
D	33	200	80	7	3600.00	10	49.72	14	47.95
D	35	200	80	8	964.56	12	59.91	16	41.18
D	40	200	80	8	2846.75	-	-	18	41.39

5 Conclusions

We have proposed a metaheuristic framework that integrates several components such as a greedy randomized adaptive procedure with biased sampling in its

construction phase and iterated greedy with a variable neighborhood descent in its local search phase. The preliminary results are very promising. The results indicate the proposed heuristic outperforms the best heuristic in terms of both solution quality and running time. The performance of the proposed approach is more robust than that of the exact method, requiring less seconds and memory to solve each instance obtaining reasonably good objective values.

Acknowledgements. 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 for providing us with the source code of her heuristic.

References

[1] Kariv, O., Hakimi, S.L.: An algorithmic approach to network location problems, Part I: The p-centers. SIAM Journal on Applied Mathematics 37, 513–538 (1979)
[2] Elloumi, S., Labbé, M., Pochet, Y.: A new formulation and resolution method for the p-center problem. INFORMS Journal on Computing 16(1), 84–94 (2004)
[3] Özsoy, F.A., Pınar, M.Ç.: An exact algorithm for the capacitated vertex p-center problem. Computers & Operations Research 33(5), 1420–1436 (2006)
[4] Albareda-Sambola, M., Díaz-García, J.A., Fernández, E.: Lagrangean duals and exact solution to the capacitated p-center problem. European Journal of Operational Research 201(1), 71–81 (2010)
[5] Scaparra, M.P., Pallottino, S., Scutellà, M.G.: Large-scale local search heuristics for the capacitated vertex p-center problem. Networks 43(4), 241–255 (2004)
[6] Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6(2), 109–133 (1995)
[7] Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (2007)
[8] Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449–467 (2001)
[9] Gendreau, M., Potvin, J.Y. (eds.): Handbook of Metaheuristics, 2nd edn. International Series in Operations Research & Management Science, vol. 146. Springer, New York (2010)
[10] Dyer, M.E., Frieze, A.: A simple heuristic for the p-center problem. Operations Research Letters 3(6), 285–288 (1985)
[11] Bresina, J.L.: Heuristic-biased stochastic sampling. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI 1996, Portland, USA, vol. 1, pp. 271–278. AAAI Press (1996)
[12] Loosemore, S., Stallman, R.M., McGrath, R., Oram, A., Drepper, U.: The GNU C Library Reference Manual: For version 2.17. Free Software Foundation, Boston, USA (2012)