Solving the *p*-Center Problem with Tabu Search and Variable Neighborhood Search

Nenad Mladenović
Service de Mathématiques de la Gestion, Université Libre de Bruxelles, Brussels, Belgium; GERAD and HEC Montréal, 3000, ch. de la Cote-Sainte-Catherine, Montreal, Quebec, Canada H3T 2A7

Martine Labbé
Service de Mathématiques de la Gestion, Université Libre de Bruxelles, Belgium

Pierre Hansen
GERAD and HEC Montréal, 3000, ch. de la Cote-Sainte-Catherine, Montreal, Quebec, Canada H3T 2A7

The *p*-Center problem consists of locating *p* facilities and assigning clients to them in order to minimize the maximum distance between a client and the facility to which he or she is allocated. In this paper, we present a basic Variable Neighborhood Search and two Tabu Search heuristics for the *p*-Center problem without the triangle inequality. Both proposed methods use the 1-interchange (or vertex substitution) neighborhood structure. We show how this neighborhood can be used even more efficiently than for solving the *p*-Median problem. Multi-start 1-interchange, Variable Neighborhood Search, Tabu Search, and a few early heuristics are compared on small- and large-scale test problems from the literature. © 2003 Wiley Periodicals, Inc.

Keywords: location; *p*-center; heuristics; tabu search; variable neighborhood search

1. INTRODUCTION

The *p*-Center problem is one of the best-known NP-hard discrete location problems [17]. It consists of locating p facilities and assigning clients to them in order to minimize the maximum distance between a client and the facility to which he or she is assigned (i.e., the closest facility). This model is used for example in locating fire stations or ambulances, where the distance from the facilities to their farthest assigned potential client should be minimum.

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of n potential locations for facilities, and $U = \{u_1, u_2, \ldots, u_m\}$, a set of m users, clients, or demand points with a nonnegative number w_i, $i = 1, \ldots, m$ (called the weight of u_i) associated with each of them. The distance between (or cost incurred for) each user–facility pair (u_i, v_j) is given as $d_{ij} = d(u_i, v_j)$. In this paper, we do not assume that the triangle inequality holds. Then, potential service is represented by a complete bipartite graph $G = (V \cup U, E)$, with $|E| = m \cdot n$. The *p*-Center problem is to find a subset $X \subseteq V$ of size p such that

$$f(X) = \max_{u_i \in U} \{w_i \min_{v_j \in X} d(u_i, v_j)\} \tag{1}$$

is minimized. The optimal value is called the *radius.* Note that, without loss of generality, we can consider the unweighted case

$$f(X) = \max_{u_i \in U} \{\min_{v_j \in X} d'(u_i, v_j)\} \tag{2}$$

by setting $d'(u_i, v_j) = w_i d(u_i, v_j)$, since the methods developed in this paper do not require the triangle inequality.

In the special case of the above model where $V = U$ is the vertex set of a complete graph $G = (V, E)$ (i.e., the so-called vertex *p*-Center problem), distances d_{ij} represent the length of the shortest path between vertices v_i and v_j and the triangle inequality is satisfied.

An integer programming formulation of the problem is the following (e.g., see [5]):

$$Minimize\ z \tag{3}$$

subject to

Received August 2000; accepted April 2003
Correspondence to: P. Hansen

$$\sum_j x_{ij} = 1, \qquad \forall\, i, \tag{4}$$

$$x_{ij} \le y_j, \qquad \forall\, i, j, \tag{5}$$

$$\sum_j y_j = p, \tag{6}$$

$$z \ge \sum_j d_{ij} x_{ij}, \qquad \forall\, i, \tag{7}$$

$$x_{ij}, y_j \in \{0, 1\}, \qquad \forall\, i, j, \tag{8}$$

where $y_j = 1$ means that a facility is located at v_j; $x_{ij} = 1$ if user u_i *is assigned to facility* v_j (and 0 otherwise). Constraints (4) express that the demand of each user must be met. Constraints (5) prevent any user from being supplied from a site with no open facility. The total number of open facilities is set to p by (6). Variable z is defined in (7) as the largest distance between a user and its closest open facility.

One way to solve the p-Center problem exactly consists of solving a series of set covering problems [19]: Choose a threshold for the radius and check whether all clients can be covered within this distance using no more than p facilities. If so, decrease the threshold; otherwise, increase it. However, this method is not efficient and no exact algorithm able to solve large instances appears to be known at present.

Classical heuristics suggested in the literature usually exploit the close relationship between the p-Center problem and another NP-hard problem called the *dominating set* problem [16, 18, 23]. Given any graph $G = (V, E)$, a *dominating set* S of G is a subset of V such that every vertex in $V \backslash S$ is adjacent to a vertex in S. The problem is to find a dominating set S with minimum cardinality. Given a solution $X = \{v_{j_1}, \ldots, v_{j_p}\}$ for the p-Center problem, there obviously exists an edge (u_i, v_{j_k}) such that $d(u_i, v_{j_k}) = f(X)$. We can delete all links of the initial problem whose distances are larger than $f(X)$. Then, X is a minimum dominating set in the resulting subgraph. If X is an optimal solution, then the subgraph with all edges of length less than or equal to $f(X)$ is called a *bottleneck graph.* Thus, to exploit the above-cited relationship, one has to search for the bottleneck graph and find its minimum dominating set. In [16] and [18], all distances are first ranked, then graphs G_t containing the t smallest edges are constructed and their domination number is found approximately (i.e., by approximating the solution of the dual problem of finding the strong stable set number). Both heuristics proposed in [16] and [18] stop when the approximated domination number reaches the value p. They differ in the order in which the subproblems are solved. No numerical results have been reported in [16], where binary search (B-S) is used until p is reached. However, the authors show that the worst-case complexity of their heuristic is $O(|E|\log|E|)$ and that, assuming the triangle inequality, the solution obtained is a 2-approximation (i.e., the objective value obtained is not larger than twice that of the optimal one). Moreover, that approximation bound is the best possible, as finding a better one would imply that P = NP. The heuristic suggested in [18] is $O(n^4)$, and only instances with up to $n = 75$ vertices are tested. Since this heuristic is based on the vertex closing principle (i.e., on the *stingy* idea), better results are obtained for large values of p than for small ones.

Of course, the three classical heuristics *Greedy* (Gr), *Alternate* (A), and *Interchange* (I) or vertex substitution, which are the most often used in solving the p-Median problem (e.g., see [12]), can easily be adapted for solving the p-Center problem.

With the Gr method, a first facility is located in such way as to minimize the maximum cost, that is, a 1-Center problem is first solved. Facilities are then added one by one until the number p is reached; each time, the location which most reduces the total cost (here, the maximum cost) is selected. In [7], a variant of Gr, where the first center is chosen at random, is suggested. In the computational results section of the present paper, results are also reported for the Gr version where all possible choices for the first center are enumerated. Such a variant will be called "Greedy Plus" (GrP).

In the first iteration of A, facilities are located at p points chosen in V, users are assigned to the closest facility, and the 1-Center problem is solved for each facility's set of users. Then, the process is iterated with the new locations of the facilities until no more changes in assignments occur. This heuristic thus consists of alternately locating the facilities and then allocating users to them—hence, its name.

Surprisingly, no results have been reported in the literature for the I procedure where a certain pattern of p facilities is given initially; then, facilities are moved iteratively, one by one, to vacant sites with the objective of reducing the total (or maximum) cost. This local search process stops when no movement of a single facility decreases the value of the objective. In the multistart version of *Interchange* (M-I), the process is repeated a given number of times and the best solution is kept. For solving the p-Median problem, the combination of Gr and I (where the Gr solution is chosen as the initial one for I) has been most often used for comparison with other newly proposed methods (e.g., [12, 25]). In our computational results section, we will do the same for the p-Center problem.

In this paper, we apply the Tabu Search (TS) and Variable Neighborhood Search (VNS) metaheuristics for solving the p-Center problem. To the best of our knowledge, no metaheuristic approach to this problem has yet been suggested in the literature. In the next section, we first propose an efficient implementation of 1-Interchange (I) (or vertex substitution) descent: One facility belonging to the current solution is replaced by another not belonging to the solution. We show how this simple neighborhood structure can be used even more efficiently than in solving the p-Median problem [12, 26]. In Section 3, we extend the I to the Chain-interchange move, as suggested in [21, 22]. In that way, a simple TS method [8–10] is obtained. In Section 4,

Algorithm Move$(c_1, c_2, d, x_{cur}, in, m, p, f, out)$

Initialization.

Set $f \leftarrow 0$, $r(x_{cur}(\ell)) \leftarrow 0$ and $z(x_{cur}(\ell)) \leftarrow 0$, for all $\ell = 1, \ldots, p$;

Add facility.

· For each user i $(i = 1, \ldots, m)$ do the following:

If $d(i, in) < d(i, c_1(i))$, then set $f \leftarrow \max\{f, d(i, in)\}$

Otherwise, update the corresponding values obtained by deleting $c_1(i)$, i.e.,

$r(c_1(i)) \leftarrow \max\{r(c_1(i)), d(i, c_1(i))\}$;

$z(c_1(i)) \leftarrow \max\{z(c_1(i)), \min\{d(i, in), d(i, c_2(i))\}\}$;

· End For

Best deletion.

· Find largest and the second largest distance if facility j is not deleted, i.e.,

$g_1 = \max\{r(x_{cur}(\ell)) | \ell = 1, \ldots, p\}$; let ℓ^* be the corresponding index;

$g_2 = \max_{\ell \neq \ell^*}\{r(x_{cur}(\ell)) | \ell = 1, \ldots, p\}$.

· Find facility *out* and value f of the best point in the neighborhood as

$f = \min\{g(\ell) | \ell = 1, \ldots, p\}$; let *out* denote the corresponding index, and

$$g(\ell) = \begin{cases} \max\{f, z(x_{cur}(\ell)), g_1\} & \textbf{if } \ell \neq \ell^*, \\ \max\{f, z(x_{cur}(\ell)), g_2\} & \textbf{if } \ell = \ell^*. \end{cases}$$

FIG. 1. Pseudocode for procedure Move.

the rules of a basic VNS are applied: A perturbed solution is obtained from the incumbent by a k-interchange neighborhood and the I descent is used to improve it. If a better solution than the incumbent is found, the search is recentered around it. In Section 5, computational results are first reported on small random instances where optimal solutions were obtained by the CPLEX solver. Based on the same computing time as a stopping condition, comparison between a few early heuristics, the M-I, the Chain interchange TS, and the VNS are reported on 40 OR-Lib test problems (graph instances devoted to testing the p-Median methods [2]). The same methods are then compared on larger problem instances taken from TSPLIB [24], with $n = 1060$ and $n = 3038$ and different values of p. Brief conclusions are drawn in Section 6.

2. VERTEX SUBSTITUTION LOCAL SEARCH

Let $X = \{v_{j_1}, \ldots, v_{j_p}\}$ denote a feasible solution of the p-Center problem. The I or *vertex substitution* neighborhood of X [noted $\mathcal{N}(X)$] is obtained by replacing, in turn, each facility belonging to the solution by each one out of it. Thus, the cardinality of $\mathcal{N}(X)$ is $p \cdot (n - p)$. The I local search heuristic, which uses it, finds the best solution $X' \in \mathcal{N}(X)$; if $f(X') < f(X)$, a move is made there $(X \leftarrow X')$, a new neighborhood is defined, and the process is repeated. Otherwise, the procedure stops, in a local minimum.

When implementing I, close attention to data structures is crucial. For example, for the p-Median problem, it is known [12, 26] that one iteration of the *Fast interchange* heuristic has a worst-case complexity $O(mn)$ [whereas a straightforward implementation is $O(mn^2)$]. We present in this section our implementation of interchange for the p-Center problem with the same low worst-case complexity.

Move Evaluation

In [12], pseudocode is given for an efficient implementation of the 1-interchange move in the context of the p-Median problem. The new facility not in the current solution X is first added, and instead of enumerating all p possible removals separately (e.g., as done in [25]), the best deletion is found during the evaluation of the objective function value (i.e., in $O(m)$ time for each added facility, see [26]). In that way, complexity of the procedure is reduced by a factor of p. The question that we address here is whether it is possible to develop an implementation with the same $O(m)$ complexity for solving the p-Center problem. The answer is positive and more details are given in the procedure *Move* below, where the objective function f is evaluated when the facility that is added (denoted with *in*) to the current solution is known, while the best one to go out (denoted with *out*) is to be found. In the description of the heuristic, we use the following notation:

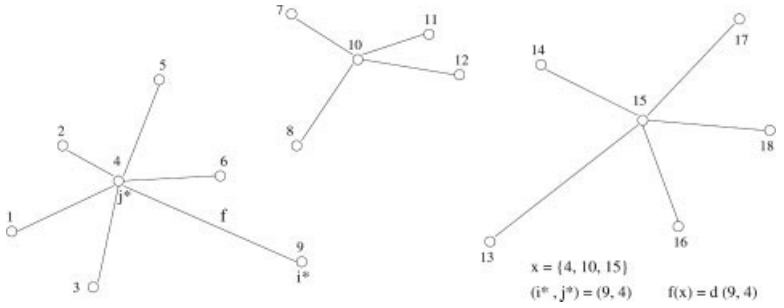

FIG. 2. Example with $m = n = 18$ and $p = 3$; the current solution is $X = \{4, 10, 15\}$.

- $d(i, j)$, distance (or cost) between user u_i and facility v_j, $i = 1, \ldots, m$; $j = 1, \ldots, n$;
- $c_1(i)$, center (closest open facility) of user u_i, $i = 1, \ldots, m$;
- $c_2(i)$, second closest open facility of user u_i, $i = 1, \ldots, m$;
- *in,* index of inserted facility (input value);
- $x_{cur}(i)$, $i = 1, \ldots, p$, current solution (indices of centers);
- $r(j)$, radius of facility v_j (currently in the solution, $j = x_{cur}(\ell)$, $\ell = 1, \ldots, p$) if it is not deleted;
- $z(j)$, largest distance between a client who was allocated to facility v_j, $j = x_{cur}(\ell)$, $(\ell = 1, \ldots, p)$, where v_j is deleted from this solution;
- f, objective function value obtained by the best interchange;
- *out,* index of the deleted facility (output value);

Since the steps of the procedure *Move* of Figure 1 are not as obvious as in the p-Median case, we give some short explanations: Note first that the current solution x_{cur} can be represented as a forest (with m edges) which consists of disconnected stars (each star being associated with a facility) and that the edge of that forest with the maximum length defines the objective function value f (see Fig. 2). Let us denote the stars by S_j, $j = 1, \ldots, p$. In the *Add facility* step, the possible new objective function value f is kept only for those clients who are attracted by the new added facility v_{in}. Beside these new distances (new assignments of clients instead of old), the new maximum distance f could be found among existing connections, but what edge would "survive" depends on the old center of the client u_i being removed or not. Fortunately, in both cases, we can store the necessary information [using arrays $r(\cdot)$ and $z(\cdot)$, as shown in Figs. 3 and 4], and without increasing the complexity, we can find the facility to be removed in the *Best deletion* step. Further details that analyze possible cases are given in Figure 1 and in the proof of Property 1.

In Figures 2–4, an example from the Euclidean plane with $n = m = 18$ and $p = 3$ is given. A facility $in = 6$ is considered to enter the current solution $X = \{4, 10, 15\}$. Facilities 8 and 9 are attracted by it (see Fig. 3). For all other users, the values of $r(4)$, $r(10)$, $r(15)$, $z(4)$, $z(10)$, and $z(15)$ are updated. [In Fig. 3, $r(10)$ and $z(10)$ are drawn for client $i = 7$; the new solution with the radius $d(1, 6)$ is shown in Fig. 4.]

From the pseudocode given in Figure 1, two properties immediately follow:

Property 1. *The worst-case complexity of the algorithm Move is $O(m)$.*

Proof. Let us denote by $X' = X \cup \{v_{in}\}$, $v_{in} \notin X$, and by X_j a solution where v_{in} is added and v_j deleted, that is,

$$X_j = X'\backslash\{v_j\} = X \cup \{v_{in}\}\backslash\{v_j\}, \quad v_j \in X.$$

Let us further denote new assignments of user u_i, $i = 1, \ldots, m$ (in each solution X_j), by $c'_1(i, j)$. Comparing two consecutive solutions X and X_j, the set of users U can be divided into three disjoint subsets:

(i) U', users attracted by a new facility v_{in} added to X (without removing any other), that is, $U' = \{u_i | d(i,$

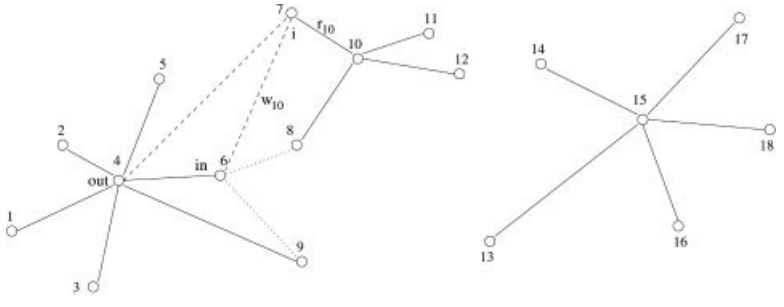

FIG. 3. Example with $m = n = 18$ and $p = 3$; facility $in = 6$ is added to the solution.

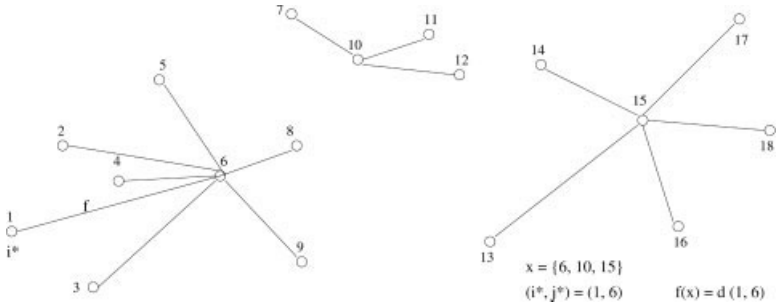

FIG. 4. Example with $m = n = 18$ and $p = 3$; the new solution is $X' = \{6, 10, 15\}$.

$in) < d(i, c_1(i))\}$, $c_1(i) \in X$, $i = 1, \ldots, m$. The new center is given by $c_1'(i, j) = in$, $u_i \in U'$. The radius of this set is

$$f' = \max_{u_i \in U'} \{d(i,in)\}. \tag{9}$$

In Figures 2 and 3, we see that $v_{in} = v_6$, $U' = \{u_8, u_9\}$, deleted edges are (v_9, v_4) and (v_8, v_{10}), and $f' = d(9, 6)$.

(ii) U_j, users that did not change their center j: $U_j = S_j \backslash U'$; that is, $c_1'(i, j) = c_1(i)$. The radius of each group of users S_j is simply

$$r(j) = \max_{u_i \in U_j} \{d(i, j) | j = c_1(i) \in X\}; \tag{10}$$

(iii) $\bar{U}_j$, users that lost their center v_j, but different from those in U': $\bar{U}_j = U \backslash (U_j \cup U')$. The new center for each such user u_i is its second closest [with index $c_2(i)$] or the new added one v_{in}: $c_1'(i, j) = \arg \min\{d(i, in), d(i, c_2(i)) | j = c_1(i)\}$. Then, the radius is (see user u_7 in Fig. 3)

$$z(j) = \max_{u_i \in \bar{U}_j} d(i, c_1'(i, j)) = \max_{u_i \in \bar{U}_j} [\min\{d(i, in), d(i, c_2(i)) | j = c_1(i) \in X\}]. \tag{11}$$

It is easy to see that f', $r(j)$ and $z(j)$ can be found in $O(m)$ time. In Figure 1, this is represented in the *Add step.* Now let us show that the *Best deletion* step is $O(p)$. For that purpose, we use the definitions of sets U', U_j, and $\bar{U}_j$ as well as relations (9), (10), and (11):

$$\begin{aligned} f(X^{(best)}) &= \min_{v_j \in X} f(X_j) = \min_{v_j \in X} [\max_{u_i \in U} d(i, c_1'(i, j))] \\ &= \min_{v_j \in X} [\max\{\max_{u_i \in U'} d(i, c_1'(i, j)), \max_{u_i \in \bar{U}_j} d(i, c_1'(i, j)), \\ &\quad \times \max_{u_i \in U_j} d(i, c_1'(i, j))] \\ &= \min_{v_j \in X} [\max\{f', z(j), \max_{\ell \neq j} r(\ell)\}] \\ &= \min_{v_j \in X} [\max\{f', z(j), r(\ell^*), \ell^* \neq j\}] \end{aligned}$$

The last expression leads to a procedure given in the *Best deletion* step of Figure 1. ∎

Property 2. *Using algorithm Move, the p different solutions from $\mathcal{N}(X)$ are visited.*

Proof. In the *Best deletion* step, p objective function values are found in array $g(\cdot)$. Those values correspond to the solutions where v_{in} is added and each facility among the p current ones is deleted in turn from x_{cur}. ∎

As mentioned before, in the procedure *Move,* it is assumed that $c_1(i)$ and $c_2(i)$ are known. The evaluation of the closest and the second-closest facilities after a move is made is the same way as for the p-Median problem [12].

Updating First- and Second-closest Facility

This routine, presented in Figure 5, is based upon the ideas of [26]. It uses for each facility a list of users for which it is the closest facility as well as a list of users for which it is the second-closest facility. It also uses arrays of closest and second-closest facilities for all users. When an ingoing vertex is considered, the possible reduction in the objective function value is computed. Then, vertices in the solution are examined one by one and it is checked whether their deletion augments the objective function value and by how much. Using the lists of users for which the vertex is closest, only $O(n)$ distances must be considered in all for that purpose. After the exchange is made, lists are updated.

Among formal variables in the description of algorithm *Update* that follows, arrays $c_1(i)$ and $c_2(i)$, $i = 1, \ldots, m$ are both input and output variables. All other formal variables in the list are input only.

The worst-case complexity of the procedure *Update* is $O(n \log n)$ when a *heap* data structure is used for updating the second closest facility $c_2(i)$.

Fast Vertex Substitution

Before giving details about the *Fast* Vertex substitution routine, we show how the complete I neighborhood $\mathcal{N}(X)$ can be increasingly reduced during the iterations, that is,

Algorithm Update $(d, goin, goout, m, p, c1, c2)$

```
For each user i (i = 1 to m) do the following
    . (* For users whose center is deleted, find new one *)
    . if c1(i) = goout then
        . if d(i, goin) ≤ d(i, c2(i)) then
            . c1(i) = goin
        . else
            . c1(i) = c2(i)
            . (* Find second closest facility for user i *)
            . find center ℓ* where d(i, ℓ) is minimum (for ℓ = 1, ..., p, ℓ ≠ c1(i));
            . c2(i) ← ℓ*
        . endif
    . else
        . if d(i, c1(i)) > d(i, goin) then
            . c2(i) ← c1(i) and c1(i) ← goin
        . else
            . if d(i, goin) < d(i, c2(i)) then
                . c2(i) = goin
            . else
                . if c2(i) = goout then
                    . find center ℓ* where d(i, ℓ) is minimum (for ℓ = 1, ..., p, ℓ ≠ c1(i));
                    . c2(i) ← ℓ*
                . endif
            . endif
        . endif
end for i
```

FIG. 5. Pseudocode for procedure update within fast 1-interchange method.

how this local search can be accelerated. However, this acceleration reduces only the constant in the heuristic's complexity, but not its worst-case behavior.

Let the current value $f(X)$ be determined by the distance between a user with index i^* and its closest facility $j^* = c_1(i^*)$ [$f = d(u_{i^*}, v_{j^*})$]. We call *critical* the user u_{i^*} who determines the objective function value. It is easy to see that there will be no improvement of f in $\mathcal{N}(X)$ if the facility to be added is farther from the critical user (with index i^*) than facility $j^* = c_1(i^*)$ (see Fig. 6). In other words, we have the following:

Property 3. *If there is a better solution in the vertex substitution neighborhood $\mathcal{N}(X)$ of the current solution X, then the new facility $v_{j'}$ must be closer to the critical user u_{i^*} than his or her previous center v_{j^*}.*

Proof. Let i' and $j' = c_1(i')$ be the new critical vertex and the new facility, respectively ($j' \notin X$). The result follows easily by contradiction. Assume that $d(u_{i^*}, v_{j'}) \geq d(u_{i^*}, v_{j^*})$. Then, the critical user u_{i^*} cannot find a facility closer than v_{j^*}. Thus, the objective function value cannot be improved, which is a contradiction. ∎

This simple fact allows us to reduce the size of the complete $p \cdot (n - p)$ neighborhood of X to $p \cdot |J(i^*)|$, where

$$J(i^*) = \{j | d(i^*, j) < f(X)\}. \tag{12}$$

Moreover, this size decreases with subsequent iterations and, thus, the speed of convergence to a local minimum increases on average. Therefore, the vertex substitution local search is more efficient in solving the p-Center problem than in solving the p-Median problem. Its pseudocode is given in Figure 7. It uses the two procedures *Move* and *Update*.

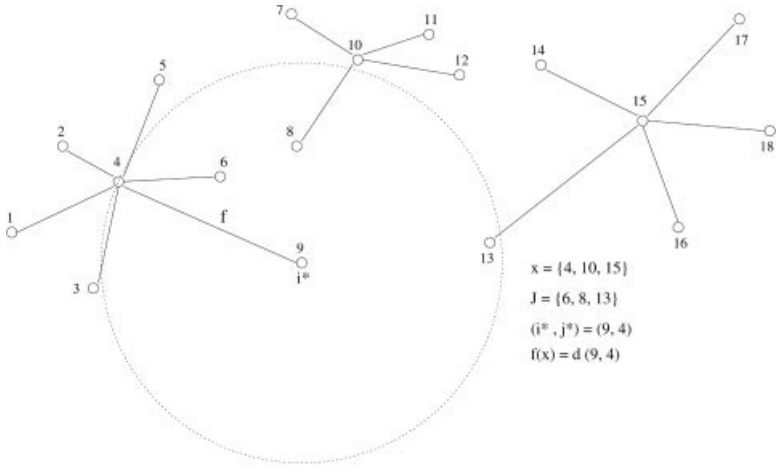

FIG. 6. Example with $m = n = 18$ and $p = 3$; if an improved solution exists in $\mathcal{N}(X)$, then only vertices inside the circle (or those from J) are candidates to enter into X (Property 3).

Regarding the data structure used by the algorithm of Figure 6, the same array $x_{opt}(j)$, $j = 1, \ldots, n$ is used to store facilities (V_{in}) in the solution (its first p elements) and out of it (V_{out}). Note that the *Update* x_{opt} step from Figure 7 uses three statements to interchange positions of two elements in the same array x_{opt}.

Property 4. *The worst-case complexity of one iteration of the algorithm I is* $O(mn)$.

Proof. This follows from the facts that procedure *Move* is used at most $n - p$ times, the complexity of *Move* is $O(m)$, and the complexity of *Update* is $O(n \log n)$. ∎

3. CHAIN SUBSTITUTION TABU SEARCH FOR THE *P*-CENTER

In this section, we suggest a basic Tabu Search (TS) [8–10] for solving the p-Center problem. The proposed heuristic exploits the similarity between the p-Center and the p-Median problems. In [12], two TS methods that use interchanges of the facilities are compared, while in [15], a third one was added in the list of compared methods. For solving the p-Center problem, we adapt one of them, that is, the TS heuristic suggested in [21, 22] that performed best, on average, on large sets of p-Median test problems from the literature, according to [12, 15]. We also describe this method in more detail than in [21, 22], adding a figure and two explicit subroutines for chain-substitution.

In [21], the vertex substitution move is extended into a so-called *chain-substitution* move where both facilities currently in the solution (V_{in}) and facilities out of it (V_{out}) are divided into non-Tabu and Tabu subsets (V^n_{in}, V^t_{in}, V^n_{out}, V^t_{out}). The interchange of two facilities is performed by changing positions of four of them in the list: The facility that goes out is chosen in V^n_{in} and replaced with one that belongs to the solution as well (i.e., from V^t_{in}), but whose turn has come to change Tabu status; in its place comes a facility from V^n_{out}, which is substituted by an element from V^t_{out}; and, finally, the chain is closed by placing the facility from V^n_{in} in V^t_{out} (see Fig. 8). In that way the TS *recency-based memory* is easily exploited, that is, the possibility of getting out of a local optimum is achieved without additional efforts.

As in the *1-Interchange* or *Vertex substitution* local search, the current solution is represented by an array x_{cur} with all facilities from V, whose first p members (V_{in}) are open. Let the current positions of the counters in the two Tabu lists V^t_{in} and V^t_{out} be denoted by cn_1 and cn_2, respectively (see Fig. 8). Then updating the new solution and the two Tabu lists can be done in four or five statements, as shown in Figure 9.

The pseudocode of CSTS-PC that uses procedures *Move* (see Fig. 1), *Update* and, *Chain-substitution* (see Fig. 9) is given in Figure 10.

In Step 1, Property 3 is used to accelerate the search (or to reduce the neighborhood) if the direction of the search is descent (i.e., if *improv = .true.* in the pseudocode). In the case of an ascent move (*improve = .false.*), the complete neighborhood is considered. Then, usual TS steps are performed: Find the best non-Tabu solution in the neighborhood, move there, and update the Tabu lists. This procedure is iterated until the stopping condition is met. CSTS-PC usually uses the lengths of the two Tabu lists t_1 and t_2 as parameters. However, in order to take advantage of the *Fast* Vertex substitution move, where the best facility to be deleted (*out*) is found, in one variant (denoted by TS-1), we fix $t_1 = 0$ to avoid Tabu status of the facility *out* [see Fig. 8(a)].

4. VARIABLE NEIGHBORHOOD SEARCH (VNS) FOR THE *P*-CENTER

VNS is a recently proposed metaheuristic for solving combinatorial and global optimization problems [13, 20]. The basic idea is a systematic change of neighborhood

Local Search Vertex Substitution (I)

Initialization

Denote a random permutation of facilities $\{1, \ldots, n\}$ by $x_{opt}(j), j = 1, \ldots, n$. Let the first p of them represent an initial solution; find the closest and second closest facility for each user i, i.e., find arrays $c_1(i), c_2(i), i = 1, \ldots, m$; find the corresponding objective function value f_{opt} and user index i^* such that $f_{opt} = d(i^*, c_1(i^*))$;

Iteration step

. $f^* \leftarrow \infty$

. For $in = x_{opt}(p+1)$ to $x_{opt}(n)$

(* Add facility in into the solution and find the best deletion *)

If $(d(i^*, in)) < d(i^*, c_1(i^*)))$ then

Run procedure $Move(c_1, c_2, d, in, m, p, f, out)$;

(* Keep the best pair of facilities to be interchanged *)

If $f < f^*$ then $f^* \leftarrow f$, $in^* \leftarrow in$, $out^* \leftarrow out$

End If

. End For

Termination.

. if $f^* \geq f_{opt}$ Stop; (* If no improvement in the neighborhood, Stop *)

Updating.

. Update the objective function value: $f_{opt} \leftarrow f^*$ and find a new i^*;

. Update x_{opt}: interchange the position of $x_{opt}(out^*)$ with that of $x_{opt}(in^*)$;

. Update the closest and second closest facilities: $Update(d, in^*, out^*, m, p, c_1, c_2)$;

. Return to *Iteration step*.

FIG. 7. Description of the fast vertex substitution or the 1-interchange (I) descent method.

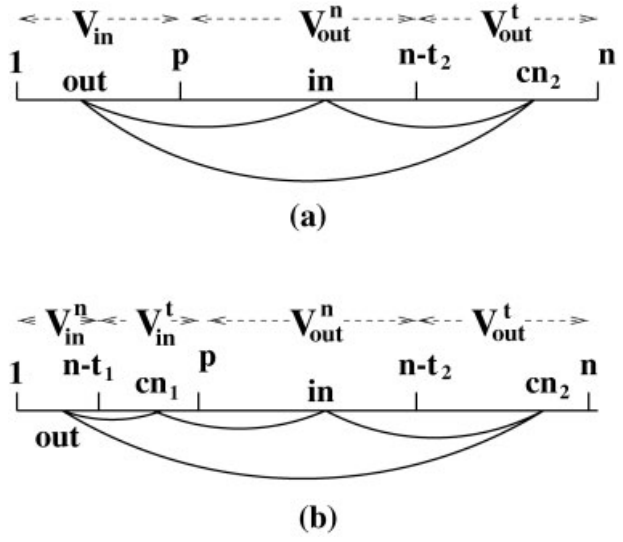

FIG. 8. Chain-substitution management: (a) TS-1: interchanges with use of one Tabu list ($V^t_{in} = \emptyset$); (b) TS-2: use of two Tabu lists.

structures within a local search algorithm. The algorithm remains centered around the same solution until another solution better than the incumbent is found and then jumps there. So, it is not a trajectory-following method such as Simulated Annealing or TS. By exploiting the empirical property of closeness of local minima that holds for most combinatorial problems, the basic VNS heuristic secures two important advantages: (i) by staying in the neighborhood of the incumbent the search is done in an attractive area of the solution space which is not, moreover, perturbed by forbidden moves; and (ii) as some of the solution attributes are already in their optimal values, local search uses several times fewer iterations than if initialized with a random solution, so, it may visit several high-quality local optima in the same CPU time one descent from a random solution takes to visit only one.

Let us denote by $\chi = \{X | X = set\ of\ p\ (out\ of\ m)\ locations\ of\ facilities\}$ a solution space of the problem. We say that the distance between two solutions X_1 and X_2 (X_1, $X_2 \in \chi$) is equal to k, if and only if they differ in k

Chain − substitution − 1(in, out, cn_2, x_{cur})	**Chain − substitution − 2**$(in, out, cn_1, cn_2, x_{cur})$
$a \leftarrow x_{cur}(out)$;	$a \leftarrow x_{cur}(out)$;
$x_{cur}(out) \leftarrow x_{cur}(cn_2)$;	$x_{cur}(out) \leftarrow x_{cur}(cn_1)$;
$x_{cur}(cn_2) \leftarrow x_{cur}(in)$;	$x_{cur}(cn_1) \leftarrow x_{cur}(in)$;
$x_{cur}(in) \leftarrow a$;	$x_{cur}(in) \leftarrow x_{cur}(cn_2)$;
	$x_{cur}(cn_2) \leftarrow a$;

FIG. 9. Chain vertex substitution with one and two Tabu lists.

Chain-Substitution Tabu Search

Initialization

(1) Find arrays x_{opt}, c_1, c_2, f_{opt} and i^* as in initialization of I (see Figure 7); (2) copy initial solution into the current one, i.e., copy f_{opt}, x_{opt}, c_1, c_2 and i^* into $f_{cur}, x_{cur}, c1_{cur}$, $c2_{cur}$ and i^*_{cur} respectively; (3) initialize Tabu lists ($V^t_{in} = V^t_{out} = \emptyset$) by using two counters $cn_1 = p$ and $cn_2 = n$; give initial values t_1 and t_2 to the Tabu list lengths; set $improve = .true.$

Repeat Main step until the stopping condition is met (e.g., $time < t_{max}$).

Main step.

(1) Reduce the neighborhood

Find facilities J to be inserted $J = \{\, j \mid d(i^*_{cur}, j) < f_{cur}\} \setminus V^t_{out}$;

if $J = \emptyset$ or $improv = .false.$, set $J = \{x_{cur}(i) | i = p+1, n\} \setminus V^t_{out}$.

(2) Find the best solution in the neighborhood

$f^* \leftarrow \infty$;

For $in \in J$, do the following:

Find facility to be deleted (out) by using procedure

Move $(c1_{cur}, c2_{cur}, d, x_{cur}, in, m, p, f_{cur}, out)$; (see Figure 1)

. Keep the best pair of facilities to be substituted

If $f < f^*$ then $f^* \leftarrow f$, $in^* \leftarrow in$, $out^* \leftarrow out$

. End For

(3) Improvement.

if $f^* < f_{opt}$ then

$f_{opt} = f^*$, $improv = .true.$; save $x_{opt}(j) = x_{cur}(j)$, $j = 1, \ldots, n$;

else set $improv = .false.$

(4) Updating.

Update objective function value: $f_{cur} \leftarrow f^*$ and find new critical user i^*;

Update x_{cur}: *Chain-substitution* $(in, out, cn_1, cn_2, x_{cur})$;

Update Tabu list counters: $cn_1 = cn_1 - 1$; $cn_2 = cn_2 - 1$; if they are

equal to $p - t_1$ or $n - t_2$, set them to p or n respectively;

Update closest and second closest facilities:

Update$(d, in^*, out^*, m, p, c_1, c_2)$;

FIG. 10. Pseudocode for CSTS-PC method.

locations. Since χ is a set of sets, a (symmetric) distance function ρ can be defined as

$$\rho(X_1, X_2) = |X_1 \backslash X_2| = |X_2 \backslash X_1|, \quad \forall X_1, X_2 \in X. \quad (13)$$

It can easily be checked that ρ is a metric function in χ; thus, χ is a metric space. The neighborhood structures that we use are induced by the metric ρ, that is, k locations of facilities ($k \leq p$) from the current solution are replaced by k others. We denote by $\mathcal{N}_k$, $k = 1, \ldots, k_{\max}$ ($k_{\max} \leq p$) the set of such neighborhood structures and by $\mathcal{N}_k(X)$ the set of solutions forming neighborhood $\mathcal{N}_k$ of a current solution X. More formally,

$$X' \in \mathcal{N}_k(X) \Leftrightarrow \rho(X', X) = k. \quad (14)$$

Another property of the p-Center problem that we must keep in mind in developing a VNS heuristic is the existence of many solutions with the same objective function value. This is especially the case for local minima. Indeed, by keeping the same critical user and its center (i.e., the same objective value), one can sometimes find many ways to select another $p - 1$ centers other than those in the current solution. For example, assume that $n = m = 300$ users are divided in $p = 3$ very distant groups A, B, and C, each having 100 users in a current solution; assume further that the critical user is in group A and that the value f is larger than the diameters of both groups B and C. Then, there are $100 \times 100 = 10{,}000$ solutions with the same value f.

The basic VNS may have difficulties escaping from the first found local optima among many with the same objective function value. Note that TS has no such difficulty since it keeps moving whether the new solution is better, equal, or worse. To overcome this in VNS, we always move to a solution of equal value, as in TS. There are other ways to escape from "plateau" solutions that we tried also: (i) to use additional criteria such as the *p-Median* and the *p-Center-sum* [11] values. The latter criterion gave better results than did the former; however, it was not better, on average, than a simple move to a solution with equal value; and (ii) to make a move with some probability (a parameter). Although we found that this parameter is closely related to p (a smaller probability is better for small p), the differences in solution value were again not significant. We conclude that there is no need to introduce additional criteria or parameters in the basic VNS version.

Pseudocode for our VNS algorithm for the p-Center problem (VNS-PC) is given in Figure 11. It uses procedures *Move* and *Update* (from Section 2).

In Step 2a, the incumbent solution X is perturbed in such a way that $\rho(X, X') = k$. As in I local search and in TS, we use here Property 3 to reduce the size of $\mathcal{N}_k(X)$. Then, X' is used as the initial solution for fast vertex substitution in Step 2b. If an equal or better solution than X is obtained, we move there and start again with small perturbations of this new best solution (i.e., $k \leftarrow 1$). Otherwise, we increase the distance between X and the new randomly generated point, that is, we set $k \leftarrow k + 1$. If k reaches $k_{\max}$ ($k_{\max}$ is the parameter of the basic version of VNS), we return to Step 1, that is, the main step iterates as long as running time is less than the maximum CPU time allowed ($t_{\max}$). Note that the point X' is generated at random in Step 2a to avoid cycling, which might occur if any deterministic rule was used.

To reduce the number of parameters, we set the value of the parameter $k_{\max}$ to p. Thus, one has to decide only how long the procedure will run, that is, $t_{\max}$. Beside usual ways to extend this basic VNS version (i.e., by introducing parameters $k_{\min}$, k_{step}, b, etc., see [13]), there are here two additional possibilities to intensify or diversify the search within the Shaking step: (i) taking a facility *in* to be inserted at random without imposing the condition that it belongs to a circle with center at the critical user and with radius f_{cur} would diversify the search; and (ii) taking the best facility to be deleted (step *Delete facility* from the procedure *Move*) would intensify the search.

5. COMPUTATIONAL RESULTS

In this section, we compare the values of the solutions and running times of several heuristics for solving the p-Center problem. Programs for all methods are coded in Fortran 77, compiled by *f77-cg89-O4 pcent.f* and run on a Sun Sparc Station 10. Since there are no benchmark test problems for the p-Center problem, we compared heuristics on instances generated at random, then on OR-Lib [2] and TSP-Lib [24] instances, devoted to the p-Median and Traveling Salesman problems, respectively. These test problems are standard ones, related to location and travel, and are easily accessible.

We first compare M-I, VNS, TS-1, and TS-2. It has been already mentioned that $k_{\max}$ (the only VNS parameter) is set to p. As explained before, TS-1 has one and TS-2 has two parameters, that is, the length(s) of their Tabu list(s). We choose variable length Tabu list options: $|V_{in}^T| = t_1 = 1 + (p - 1) \cdot Rnd$ and $|V_{out}^T| = t_2 = 1 + (n - p - 1) \cdot Rnd$, where *Rnd* is a uniformly distributed random number from the interval (0, 1).

Small Random Instances

First, we wanted to determine the largest problem that can be solved exactly in reasonable time by the latest version of the well-known CPLEX code (i.e., CPLEX Linear Optimizer 6.0 with Mixed Integer and Barrier Solvers), and how good a lower bound is found by solving the linear program obtained when the integer constraints (8) are relaxed. Test problems are constructed as follows: Points are generated at random in a square [0, 100] × [0, 100] and Euclidean distances provide the initial matrix D; it is assumed that $m = n$. In columns 1 and 2 of Table 1, parameters of the problem are given; column 3 contains the optimal values obtained by CPLEX; the value of the LP

Variable Neighborhood Search

Initialization

(1) Find arrays x_{opt}, c_1 and c_2 and f_{opt} and i^* as in initialization of I (see Figure 6); (2) the set of neighborhood structures $\mathcal{N}_k$ ($k = 1, \ldots, k_{max}$) is induced by the distance function ρ (see (9) and (10)); (3) copy initial solution into the current one, i.e., copy f_{opt}, x_{opt}, c_1, c_2 and i^* into $f_{cur}, x_{cur}, c1_{cur}$, $c2_{cur}$ and i^*_{cur} respectively.

Repeat Main step until the stopping condition is met (e.g., $time < t_{max}$).

Main step.

. (1) $k \leftarrow 1$;

. (2) Until $k = k_{max}$, repeat the following steps:

(2a) Shaking operator

(* Generate a solution at random from the k^{th} neighborhood *)

For $j = 1$ to k, do the following:

. Take facility to be inserted (in) at random, if it satisfies $d(i^*_{cur}, in) < f_{cur}$;

. Find facility to be deleted (out) at random;

. Find $c1_{cur}$ and $c2_{cur}$ for such interchange, i.e., run subroutine

Update $(d, in, out, m, p, c1_{cur}, c2_{cur})$;

. Update x_{cur}, f_{cur} and i^*_{cur} accordingly;

End For

(2b) Local search.

Apply algorithm I (without *Initialization* step), with x_{cur}, $c1_{cur}$, $c2_{cur}$ and i^*_{cur} as input and output values; denote the corresponding objective function value by f_{cur} (see Figure 6);

(2c) Move or not.

If $f_{cur} \leq f_{opt}$ then

(* Save current solution as the incumbent; return to $\mathcal{N}_1$ *)

$f_{opt} \leftarrow f_{cur}$; $x_{opt} \leftarrow x_{cur}$; $c_1 \leftarrow c1_{cur}$; $c_2 \leftarrow c2_{cur}$; $i^* \leftarrow i^*_{cur}$ and set $k \leftarrow 1$

else

(* Current solution is the incumbent one; change the neighborhood *)

$f_{cur} \leftarrow f_{opt}$; $x_{cur} \leftarrow x_{opt}$; $c1_{cur} \leftarrow c_1$; $c2_{cur} \leftarrow c_2$; ; $i^*_{cur} \leftarrow i^*$ and set $k \leftarrow k + 1$

End if

End For

FIG. 11. Pseudocode for VNS-PC method.

solution and % gap are given in columns 4 and 5, respectively, where % gap is calculated as

$$\frac{f_{ex} - f_{lp}}{f_{ex}} \times 100\%. \tag{15}$$

The next three columns report CPLEX outputs: Running time, number of iterations, and number of search tree nodes. The best values obtained with the three metaheuristics proposed in this paper (i.e., M-I, VNS, and TS) and B-S [16] are given in columns 9–12.

It appears that (i) one has to wait almost 8 hours to get the exact solution (f_{ex}) when $n = m = 40$ and $p = 10$; (ii) all three metaheuristics proposed in this paper (i.e., M-I,

TABLE 1. Exact LP relaxed, and heuristic values for small instances.

					Inter "mipopt" CPLEX solver			Objective values			
n	p	f_{ex}	f_{lp}	% Gap	Time	# Iter.	# Nodes	M-I	VNS	TS	B-S
30	5	28.30	16.98	40.00	88.68	60,357	2652	28.30	28.30	28.30	41.16
	10	20.04	8.39	58.13	446.89	388,483	42,664	20.04	20.04	20.04	21.88
	15	12.24	4.38	64.22	247.28	156,416	25,312	12.24	12.24	12.24	17.12
40	5	30.10	17.69	41.23	1111.38	494,122	15,060	30.10	30.10	30.10	54.85
	10	20.04	9.70	51.60	28,517.64	18,061,725	1,247,390	20.04	20.04	20.04	28.30

VNS, and TS) solved exactly all problems from Table 1 almost immediately (in 0.01 seconds at most); (iii) the existing B-S heuristic did not solve any test problem exactly; and (iv) % gap increases with the value of p.

OR-Lib Test Problems

As mentioned above, the 40 test problems used in this section for comparing the p-Center heuristics were originally designed for testing the p-Median problem [2]. In all of them, the set of facilities is equal to the set of users ($m = n$). The problem parameters range from instances with $n = 100$ nodes and $p = 5$, 10, 20, and 33 up to instances with $n = 900$ and $p = 5$, 10, 90. To get the matrix D that is used by our heuristics, an *all shortest paths* algorithm is first run [the CPU time for this $O(n^3)$ procedure is not included in the tables below].

In Table 2, results for the first 12 OR-Lib problems obtained by CPLEX are reported. In columns 2 and 3, parameters of the problems are given; values of best-known solutions (obtained by long runs of the heuristics suggested in this paper) are reported in column 4; in column 5, the optimal value of the LP relaxation is given, while in column 6, gaps calculated by (15) are presented (where f_{ex} is replaced by the best-known value); and the last two columns give characteristics of LP solutions. To get the LP solutions, we use the primal method, that is, the *primopt* command.

TABLE 2. The p-Center problem tested on the first 12 OR-Lib test instances; gaps are calculated with respect to the best-known solution.

				LP Relaxation			
Pr. no.	n	p	Best known	f_{lp}	% Gap	Time	# Iter.
1	100	5	127	90.92	28.41	101.18	7029
2	100	10	98	63.35	35.36	80.64	5084
3	100	10	93	62.48	32.82	129.16	6929
4	100	20	74	41.50	43.92	71.08	3911
5	100	33	48	19.12	60.17	45.25	2832
6	200	5	84	62.88	25.15	4797.54	62,521
7	200	10	64	45.10	29.53	2251.59	31,013
8	200	20	55	33.84	38.48	2096.65	27,987
9	200	40	37	20.02	45.88	1247.49	15,027
10	200	67	20	8.76	56.22	786.27	11,382
11	300	5	59	45.93	22.15	4551.31	68,360
12	300	10	51	38.41	24.69	9275.09	81,323

From Table 2, it appears that (i) % gaps are somewhat smaller than in random instances reported in Table 1. They vary between 25 and 60% (while in much smaller random instances, % gaps were between 40 and 64%). (ii) Since the number of variables for LP is $O(n^2)$, LP computing times are large. For example, a lower bound for $n = m = 200$ and $p = 5$ is found after more than 1 hour of processor time and about 50,000 Simplex iterations. However, the best-known heuristic solution is obtained in less than 2 seconds by all three new heuristics (see Table 4). (iii) The p-Center problem is much harder to solve exactly than is the p-Median problem. We tried to get the exact solution for problem 1 using CPLEX, but after 24 hours of computing time, the gap was reduced only by one-half.

We then tested several classical heuristics, briefly described in the Introduction: Binary search (B-S) [16]; 1-Interchange (I), suggested in this paper; Greedy (Gr), Greedy Plus (GrP), and Alternate [6] (A). Beside these three local searches, we tested Gr + I, GrP + I, A + I, Multistart Interchange (M-I), Multistart Alternate (M-A), and Multistart A + I (M-A + I). In column 4 of Table 3, values of the best-known solutions are reported, followed by values obtained by different descent methods. The maximum time allowed for the Multistart approach, reported in the last three columns of Table 3, was set to $2n$ seconds. The average % errors, the average CPU times, and the number of instances whose solution is the same as the best-known are given in the last three rows of Table 3. It appears that

(i) None of the classical heuristics gave satisfactory results; the average % deviations of B-S, I, Gr, Gr + I, GrP, GrP + I, A, and A + I are 48.5, 62.4, 119.7, 90.1, 81.9, 67.0, 94.0, and 62.9%, respectively;

(ii) The B-S method outperforms the other methods, on average, but it runs more than 400 times longer than does I, A, or A + I, more than 10 times longer than Gr + I (compare 43.9, 0.1, and 3.8 seconds given at the *Average time* line of the Table 3). B-S is worse than the other local searches for instances with small p and better for large n and p;

(iii) Gr + I found the best-known solution only once ($n = 400$ and $p = 5$), while for the p-Median problem (as reported in [25]), Gr + I found the optimal solution 17 times on the same 40 test instances. This is one more indicator of how difficult the p-Center problem is;

TABLE 3. Comparison of classical heuristics on 40 OR-Lib test instances.

				Objective values										
Pr. no.	n	p	Best known	B-S	I	Gr	Gr + I	GrP	GrP + I	A	A + I	M-I	M-A	M-A + I
1	100	5	127	184	133	166	133	148	133	148	148	127	127	127
2	100	10	98	142	102	155	109	131	119	112	112	98	98	98
3	100	10	93	137	108	191	102	168	102	131	99	93	95	93
4	100	20	74	109	135	157	111	127	111	111	111	74	79	74
5	100	33	48	76	74	143	90	106	70	87	87	48	59	48
6	200	5	84	113	93	115	100	99	94	100	92	84	84	84
7	200	10	64	87	77	97	84	85	84	77	77	64	66	64
8	200	20	55	76	92	110	84	94	94	86	86	58	67	57
9	200	40	37	59	57	98	68	76	73	71	71	46	56	42
10	200	67	20	33	53	70	70	70	70	77	42	30	34	29
11	300	5	59	71	68	71	66	68	68	64	64	59	59	59
12	300	10	51	78	62	83	69	77	77	77	59	51	72	51
13	300	30	36	58	54	72	59	58	55	59	54	41	41	39
14	300	60	26	43	60	76	76	60	60	76	76	36	40	35
15	300	100	18	30	40	59	51	49	49	44	44	29	35	27
16	400	5	47	55	59	61	47	54	47	52	48	47	47	47
17	400	10	39	55	53	53	50	51	51	49	43	40	40	39
18	400	40	28	44	50	62	50	61	50	50	50	38	40	34
19	400	80	19	31	36	41	41	38	38	36	36	28	28	26
20	400	133	14	22	37	49	49	34	32	44	44	26	29	22
21	500	5	40	52	43	52	43	47	43	45	43	40	40	40
22	500	10	38	56	44	63	47	53	47	53	42	40	42	39
23	500	50	23	34	35	45	45	38	37	34	34	30	29	28
24	500	100	16	24	33	46	31	39	39	39	39	25	30	22
25	500	167	12	20	30	47	40	30	30	47	47	22	39	20
26	600	5	38	50	40	51	40	44	40	46	39	38	38	38
27	600	10	32	43	40	46	39	39	39	39	39	33	34	33
28	600	60	19	30	33	57	57	39	34	57	27	25	57	22
29	600	120	13	22	36	38	36	36	36	36	23	23	36	19
30	600	200	11	16	28	40	40	37	29	30	30	20	20	20
31	700	5	30	38	35	39	32	37	32	35	33	30	30	30
32	700	10	29	44	39	72	72	44	35	72	33	30	72	30
33	700	70	16	25	30	32	29	30	30	36	36	22	21	20
34	700	140	12	20	28	45	31	31	29	41	26	21	41	18
35	800	5	30	37	36	38	31	34	33	36	34	30	30	30
36	800	10	27	43	31	42	42	40	33	42	31	28	42	28
37	800	80	16	24	30	33	33	32	26	40	25	23	22	20
38	900	5	29	38	31	40	40	40	40	40	30	29	40	29
39	900	10	23	37	27	74	74	38	28	74	28	24	74	24
40	900	90	14	21	27	30	28	25	25	22	22	21	20	19
Average time				43.9	0.1	3.6	3.8	1606.5	1606.5	0.1	0.1	163.6	120.5	150.0
Average % dev.			0.0	48.5	62.4	119.7	90.1	81.9	67.0	94.0	62.9	23.9	55.4	17.4
No. solved			40	0	0	0	1	0	1	0	0	15	9	16

(iv) In 12 instances, Gr solutions were local minima with respect to the vertex substitution neighborhood;
(v) Gr + I improves the Gr solution more than does GrP even if the later spends n times more CPU time (compare average times of 3.8 and 1606.5 for Gr + I and GrP, respectively);
(vi) The I heuristic is able to improve solutions obtained by others within a small amount of additional CPU times. However, regarding both solution quality and computing times, the best choice seems to be A + I;
(vii) M-I is significantly better than is M-A. Use of two neighborhood structures A and I is the best choice in the Multistart approach reported in last three columns (compare average % errors of 23.9, 55.4, and 17.4% for M-I, M-A, and M-A + I, respectively).

The new heuristics proposed in this paper were also compared on all 40 OR-Lib test problems and the results are given in Table 4. For each test problem, methods are allowed to spend the same computing time $t_{\max}$. For the results presented in Table 4, $t_{\max} = 2n$ was used. The values of the best-known solutions, which we found in our experiments, are reported in the fourth column, and the next four columns report the values obtained by each heuristic; columns 8–12 give % deviation of the objective values (with respect to the best-known solution from column 4), while the next four columns report CPU times, in seconds, spent by the heuristics to get their best solution.

In ech test, the same initial solution for TS-1, TS-2, and VNS is generated: p facilities are chosen at random, fol-

TABLE 4. OR-Lib test instances with $t_{max} = 2 \cdot n$ seconds.

				Objective values				% Deviation				Time (seconds)			
Pr. no.	n	p	Best known	M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1	TS-2
1	100	5	127	127	127	127	127	0.00	0.00	0.00	0.00	0.03	0.04	0.00	0.04
2	100	10	98	98	98	98	98	0.00	0.00	0.00	0.00	10.00	4.45	0.39	0.04
3	100	10	93	93	93	93	93	0.00	0.00	0.00	0.00	26.13	0.17	11.54	10.54
4	100	20	74	74	74	74	74	0.00	0.00	0.00	0.00	119.75	0.30	6.03	3.38
5	100	33	48	48	48	48	48	0.00	0.00	0.00	0.00	10.92	0.11	0.13	0.03
6	200	5	84	84	84	84	84	0.00	0.00	0.00	0.00	0.41	1.51	0.67	0.02
7	200	10	64	64	64	64	64	0.00	0.00	0.00	0.00	7.10	1.30	0.98	0.12
8	200	20	55	58	55	55	55	5.45	0.00	0.00	0.00	89.14	2.22	1.05	1.87
9	200	40	37	46	37	37	37	24.32	0.00	0.00	0.00	24.89	10.89	3.80	2.20
10	200	67	20	30	20	20	20	50.00	0.00	0.00	0.00	87.32	9.04	18.00	12.21
11	300	5	59	59	59	59	59	0.00	0.00	0.00	0.00	4.38	1.52	2.02	3.89
12	300	10	51	51	51	51	51	0.00	0.00	0.00	0.00	569.93	5.39	1.38	11.31
13	300	30	36	41	36	36	36	13.89	0.00	0.00	0.00	52.19	10.21	145.77	1.35
14	300	60	26	36	26	26	26	38.46	0.00	0.00	0.00	165.72	70.67	104.63	38.80
15	300	100	18	29	18	25	18	61.11	0.00	38.89	0.00	550.08	55.86	7.75	6.30
16	400	5	47	47	47	47	47	0.00	0.00	0.00	0.00	2.12	0.08	0.05	0.00
17	400	10	39	40	39	39	39	2.56	0.00	0.00	0.00	142.79	26.00	1.50	0.25
18	400	28	38	29	28	28	35.71	3.57	0.00	0.00	8.63	54.69	119.61	19.33	
19	400	80	19	28	19	23	19	47.37	0.00	21.05	0.00	447.76	300.11	21.95	121.06
20	400	133	14	26	14	22	14	85.71	0.00	57.14	0.00	631.47	218.61	148.80	52.25
21	500	5	40	40	40	40	40	0.00	0.00	0.00	0.00	2.91	1.24	0.11	0.03
22	500	10	38	40	38	38	38	5.26	0.00	0.00	0.00	111.76	82.63	46.39	35.75
23	500	50	23	30	23	23	23	30.43	0.00	0.00	0.00	87.47	112.33	14.71	8.30
24	500	100	16	25	16	18	16	56.25	0.00	12.50	0.00	12.09	264.46	246.47	27.54
25	500	167	12	22	12	24	12	83.33	0.00	100.00	0.00	30.39	175.67	0.32	226.76
26	600	5	38	38	38	38	38	0.00	0.00	0.00	0.00	0.60	0.58	2.28	0.09
27	600	10	32	33	32	32	32	3.12	0.00	0.00	0.00	78.53	5.07	0.30	0.85
28	600	60	19	25	19	19	19	31.58	0.00	0.00	0.00	230.58	25.09	764.76	87.97
29	600	120	13	23	13	23	13	76.92	0.00	76.92	0.00	9.30	762.44	0.08	151.28
30	600	200	11	20	11	19	11	81.82	0.00	72.73	0.00	835.73	196.95	0.10	84.65
31	700	5	30	30	30	30	30	0.00	0.00	0.00	0.00	34.57	0.58	0.25	0.13
32	700	10	29	30	29	29	29	3.45	0.00	0.00	0.00	117.41	165.26	27.07	84.60
33	700	70	16	22	16	16	16	37.50	0.00	0.00	0.00	497.52	806.77	491.52	96.08
34	700	140	12	21	12	20	12	75.00	0.00	66.67	0.00	52.79	160.15	0.10	80.81
35	800	5	30	30	30	30	30	0.00	0.00	0.00	0.00	15.57	6.67	1.85	13.17
36	800	10	27	28	27	27	27	3.70	0.00	0.00	0.00	82.44	105.99	58.88	50.60
37	800	80	16	23	16	22	16	43.75	0.00	37.50	0.00	78.19	1197.86	0.33	99.00
38	900	5	29	29	29	29	29	0.00	0.00	0.00	0.00	3.50	1.92	0.45	0.81
39	900	10	23	24	24	23	24	4.35	4.35	0.00	4.35	1297.79	5.98	1401.01	5.50
40	900	90	14	21	14	22	14	50.00	0.00	57.14	0.00	10.46	493.79	0.09	47.74
Average								23.86	0.20	13.59	0.11	163.56	133.67	91.30	30.11

lowed by the I local search. Each instance is solved by each method only once. The following observations can be derived from Table 4:

(i) All three new methods outperform the constructive heuristic B-S;

(ii) TS-2 and VNS perform best on average. The total average % deviations were 0.18, 0.20, 13.59, and 23.86% for TS-2, VNS, TS-1, and M-I;

(iii) M-I was not successful in solving problems with a large number of local minima, that is, for large p. We suspect that M-I suffers from the *central limit catastrophe* [1], since this was observed earlier for some other combinatorial problems (see, e.g., [3] for Traveling Salesman and Graph Bisection problems, [4] for the Multi-source Weber problem, [12] and [14] for the p-Median and for the minimum sum of squares clustering, respectively): When problems grow large, random local minima drawn from an enormous population are almost surely of "average" quality and increasing the number of restarts does not help (almost surely). To check if this holds for M-I in solving the p-Center problem, we allowed much more time for instances with large p and only a few times was the solution slightly improved. For example, we ran problem no. 20 (where $n = 400$ and $p = 167$) for 10,000 seconds, and the % deviation was reduced from 85.71 to 78.57%, that is, the value was reduced from 26 to 25;

(iv) The maximum time allowed was not always properly used. For example, in solving problem 40 by TS-1, the best solution was found after 0.09 seconds although its % error was 57% and t_{max} was 1800 seconds. This leads to the conclusion that better estimation of parameters is needed for both VNS and TS;

TABLE 5. Average results for 40 OR-Lib test instances for t_{max}.

	% Deviation				Average times (seconds)					# of problems solved			
t_{max} (seconds)	M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1	TS-2	t_{max}	M-I	VNS	TS-1	TS-2
n/10,000	55.55	49.05	37.11	66.36	0.08	0.07	0.03	0.02	0.05	0	0	1	2
n/5000	51.44	44.22	31.67	57.93	0.11	0.09	0.05	0.04	0.09	0	1	2	4
n/1000	41.84	30.44	25.98	22.43	0.22	0.28	0.17	0.28	0.46	2	3	6	11
n/500	36.34	23.26	24.51	16.27	0.49	0.58	0.24	0.46	0.92	2	6	7	12
n/100	32.41	12.16	20.57	7.69	1.67	2.26	0.88	4.41	4.60	6	14	16	15
n/50	28.94	8.61	19.65	7.00	4.10	4.51	1.57	2.46	9.20	7	16	17	19
n/20	27.14	4.55	17.98	3.37	8.05	7.52	3.21	5.83	23.00	10	20	19	24
n/10	26.23	3.61	17.03	2.38	12.91	13.98	8.40	10.86	46.00	10	22	23	28
n	24.19	0.70	13.37	0.18	63.42	71.97	40.63	33.24	460.00	13	35	28	38
$2 \cdot n$	23.86	0.20	13.59	0.11	163.56	133.67	91.30	33.24	920.00	15	38	31	39

(v) TS-1 was developed in order to exploit Observation 1 made above in full. In other words, we wanted to prevent situations where some good move had been made Tabu. However, although in the first stage of the search it looks very efficient, after some time, TS-1 becomes over intensified and cannot escape easily from deep local optima.

Some observations above can be confirmed from the results of Table 5, where average results over all 40 test problems for different values of t_{max} are reported: From $t_{max} = n/10{,}000$ seconds per problem to $t_{max} = 2n$ seconds. (Each line of Table 5 gives average results, as the last line in Table 4, but for the different t_{max}). The last three columns of Table 5 give the number of problems (out of 40) for which each method got the best-known solution. The following observations could be derived:

(i) If small running time is allowed, TS-1 performs best. If $t_{max} = n/10{,}000$ (which is 0.05 seconds on average for all 40 problems), its average % deviation is around 37%, while VNS, M-I, and TS-2 yield 49, 55, and 66%, respectively;
(ii) Parameterless VNS systematically decreases the gap; finally, it found best-known solutions in 38 among 40 instances;
(iii) The behavior of TS-2 is interesting: If a small running time is allowed, it performs worst. Finally, it found 39 best-known solutions, faster than VNS; this suggest to start the search with one Tabu list strategy (i.e., by keeping parameter $t_1 = 0$) and continue the search with two Tabu lists;
(iv) It appears that M-I, VNS, TS-1, and TS-2 reached best-known values in 15, 38, 31, and 39 cases, respectively.

TSP-Lib Test Problem

The larger problem instances studied are taken from TSP-Lib [24]. The first one consists of 1060 and the second 3038 points in the plane. In the comparison of methods, the classical heuristic B-S is also included in Table 6 to check

TABLE 6. The p-Center results for the $n = 1060$ TSP-Lib test problem; maximum time allowed for each instance is set to 500 seconds on a SUN Sparc 10.

		Objective values				% Deviation				Time (seconds)			
p	Best known	M-I	VNS	TS-1	B-S	M-I	VNS	TS-1	B-S	M-I	VNS	TS-1	B-S
10	2273.08	2360.71	2280.09	2273.08	3291.10	3.86	0.31	0.00	44.79	363.62	94.93	23.43	19.48
20	1594.87	1650.34	1611.95	1611.92	2486.17	3.48	1.07	1.07	55.89	68.36	20.49	480.43	44.06
30	1217.48	1304.87	1220.41	1217.48	1914.30	7.18	0.24	0.00	57.24	365.92	373.46	351.57	67.19
40	1020.56	1105.40	1050.45	1051.74	1645.72	8.31	2.93	3.06	61.26	226.09	279.75	194.06	118.13
50	922.11	950.65	922.14	922.11	1393.79	3.10	0.00	0.00	51.15	39.79	477.18	333.72	129.06
60	781.17	862.56	806.52	791.08	1290.05	10.42	3.25	1.27	65.14	50.50	446.89	254.87	142.82
70	710.76	790.40	721.37	727.59	1117.59	11.21	1.49	2.37	57.24	6.34	422.73	369.04	217.57
80	652.21	727.59	670.53	720.93	999.84	11.56	2.81	10.54	53.30	112.95	398.84	54.29	276.61
90	607.88	700.55	640.23	636.42	999.84	15.24	5.32	4.70	64.48	218.86	111.08	488.33	364.50
100	570.01	640.23	582.92	583.32	848.59	12.32	2.26	2.34	48.87	214.29	430.33	457.49	259.31
110	538.84	604.44	565.72	570.18	806.52	12.17	4.99	5.82	49.68	235.05	186.60	465.06	279.34
120	510.28	582.90	551.90	538.74	721.37	14.23	8.16	5.58	41.37	304.66	218.84	357.75	292.51
130	499.65	582.95	500.14	538.27	728.55	16.67	0.10	7.73	45.81	441.44	473.65	469.60	375.92
140	453.13	552.76	500.12	499.96	707.66	21.99	10.37	10.33	56.17	316.09	214.06	280.17	550.14
150	447.01	538.84	453.16	495.02	667.55	20.54	1.38	10.74	49.34	477.56	428.16	440.86	785.94
Average					10.77	2.79	4.10	50.11	215.10	286.06	313.83	245.16	

TABLE 7. The p-Center results for the $n = 3038$ TSP-Lib test problem; maximum time allowed for each instance is set to 1000 seconds on a SUN Sparc 10.

		Objective values				% Deviation				Time (seconds)			
p	Best known	M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1	TS-2
50	307.48	329.66	317.00	307.48	307.88	7.21	3.10	0.00	0.13	762.15	578.81	981.91	315.24
100	215.67	234.10	220.06	219.59	215.67	8.55	2.04	1.82	0.00	994.28	570.60	587.98	683.49
150	174.83	195.21	174.83	174.83	177.23	11.66	0.00	0.00	1.37	602.95	52.99	984.27	895.88
200	157.00	181.18	157.88	157.00	157.09	15.40	0.56	0.00	0.06	238.56	747.00	661.67	903.46
250	137.54	165.47	140.98	137.54	138.92	20.31	2.50	0.00	1.00	640.69	103.72	952.57	998.18
300	123.33	158.38	123.33	124.60	123.44	28.42	0.00	1.03	0.09	985.02	786.96	892.71	893.49
350	118.02	146.82	118.02	118.83	118.07	24.40	0.00	0.69	0.04	354.46	264.59	783.24	992.13
400	107.65	137.57	107.65	114.11	113.22	27.79	0.00	6.00	5.17	723.01	904.88	518.61	836.23
450	101.51	135.09	101.51	101.98	102.14	33.08	0.00	0.46	0.62	267.58	674.55	973.90	757.54
500	94.37	124.33	94.37	97.20	96.14	31.75	0.00	3.00	1.88	351.76	937.70	732.09	791.76
Average						20.86	0.82	1.30	1.04	592.05	562.18	806.89	806.74

if it works better for larger problems. Results for TS-2 are not reported for the 1060 problem since they were slightly worse than those of TS-1 (the average % deviation for different p was 4.26% and the average time was 316.20 seconds compared with 4.10% and 313.83 seconds for TS-1).

B-S was excluded again from Table 7 since it was clear from Table 6 that it cannot compete with the other heuristics.

It appears that (i) VNS and TS perform better than does M-I; (ii) TS methods perform better than VNS for smaller values of p, and the opposite is true for larger p, although VNS is slightly better on average; and (iii) TS-1 and TS-2 perform similarly in larger problem instances.

6. CONCLUSIONS

For the first time, metaheuristic approaches are suggested as a means for solving the p-Center problem, one of the basic models in discrete location theory. This problem consists of finding p facilities and assigning m clients to them such that the maximum distance (or cost) between a client and the facility to which he or she is allocated is minimum. The p-Center problem has numerous applications, such as the location of fire or ambulance stations in urban areas. However, no satisfactory exact or heuristic method has been offered up to now for solving large instances.

The p-Center problem is harder to solve than is another basic discrete location problem, the p-Median problem. While for the latter numerous exact and heuristic methods have been suggested, for the former, there are only a few, the last one being suggested more than 10 years ago. The p-Median instances can be easily solved exactly with up to 500 clients, while for the p-Center problem the largest instance considered has 75 clients and was solved heuristically.

However, we showed in this paper that the state-of-the-art heuristics for the p-Median problem could be adapted for solving the p-Center problem. Moreover, the fast implementation of the I (or vertex substitution) local search is even faster when applied to the p-Center problem (although with the same theoretical worst-case bound). This fast local search is then used in constructing three metaheuristics: Multistart local search, TS, and VNS. Since benchmark test problems do not exist for the p-Center problem, we used the same test instances as for the p-Median problem, that is, the 40 OR-Lib problems. Similar conclusions as in solving the p-Median problem are obtained: (i) All three heuristics significantly outperform the previous "B-S" heuristic (which, assuming the triangle inequality, has the best theoretical bound, i.e., for which the objective function value is at most twice the optimal one); (ii) VNS and TS outperform M-I; and (iii) VNS is, on average, better than is TS, although TS performs slightly better for small p.

Computational results reported in this paper were obtained by parameter free versions of M-I, VNS, and TS. Future work can consider variants with easy to use parameters. Moreover, for very large problem instances, it should be investigated whether the variable neighborhood decomposition search method [15] would be as successful or not in solving the p-Center problem as it is in solving the p-Median problem.

REFERENCES

[1] E.B. Baum, "Toward practical 'neural' computation for combinatorial optimization problems," Neural networks for computing, J. Denker (Editor), American Institute of Physics, College Park, MD, 1986.

[2] J.E. Beasley, A note on solving large p-Median problems, Eur J Oper Res 21 (1985), 270–273.

[3] K.D. Boese, A.B. Kahng, and S. Muddu, A new adaptive multi-start technique for combinatorial global optimizations, Oper Res Lett 16 (1994), 101–113.

[4] J. Brimberg, P. Hansen, N. Mladenović, and É. Taillard, Improvements and comparison of heuristics for solving the multisource Weber problem, Oper Res 48 (2000), 444–460.

[5] M. Daskin, Network and discrete location, Wiley, New York, 1995.

[6] Z. Drezner, The p-Center problem—heuristics and optimal algorithms, J Oper Res Soc 35 (1984), 741–748.

[7] M.E. Dyer and A.M. Frieze, A simple heuristic for the p-Center problem, Oper Res Lett 3 (1985), 285–288.

[8] F. Glover, Tabu search—Part I, ORSA J Comput 1 (1989), 190–206.

[9] F. Glover, Tabu search—Part II, ORSA J Comput 2 (1990), 4–32.

[10] P. Hansen and B. Jaumard, Algorithms for the maximum satisfiability problem, Computing 44 (1990), 279–303.

[11] P. Hansen, M. Labbé, and M. Minoux, The p-Center-sum location problem, Cah CERO 36 (1994), 203–219.

[12] P. Hansen and N. Mladenović, Variable neighborhood search for the p-Median, Locat Sci 5 (1997), 207–226.

[13] P. Hansen and N. Mladenović, "An introduction to variable neighborhood search," Metaheuristics, advances and trends in local search paradigms for optimization, S. Voss, S. Martello, I.H. Osman and C. Roucairol. (Editors), Kluwer, Dordrecht, 1999, pp. 433–458.

[14] P. Hansen and N. Mladenović, J-MEANS: A new local search heuristic for minimum sum-of-squares clustering, Pattern Recog 34 (2001), 405–413.

[15] P. Hansen, N. Mladenović, and D. Perez-Brito, Variable neighborhood decomposition search, J Heuristics 7 (2001), 335–350.

[16] D. Hochbaum and D. Shmoys, A best possible heuristic for the k-Center problem, Math Oper Res 10 (1985), 180–184.

[17] O. Kariv and S.L. Hakimi, An algorithmic approach to network location problems. Part 1: The p-Centers, SIAM J Appl Math 37 (1979), 513–538.

[18] J. Martinich, A vertex-closing approach to the p-Center problem, Naval Res Log 35 (1988), 185–201.

[19] E. Minieka, The m-Center problem, SIAM Rev 12 (1970), 138–139.

[20] N. Mladenović and P. Hansen, Variable neighborhood search, Comp Oper Res 24 (1997), 1097–1100.

[21] N. Mladenović, J.P. Moreno, and J. Moreno-Vega, Tabu search in solving the p-facility location-allocation problems, Les Cahiers du GERAD, G-95-38, Montreal, 1995.

[22] N. Mladenović, J.P. Moreno, and J. Moreno-Vega, A chain-interchange heuristic method, Yugoslav J Oper Res 6 (1996), 41–54.

[23] J. Plesnik, A heuristic for the p-Center problem in graphs, Discrete Appl Math 17 (1987), 263–268.

[24] G. Reinelt, TSP-Lib-A traveling salesman library, ORSA J Comput 3 (1991), 376–384.

[25] S. Voss, A reverse elimination approach for the p-Median problem, Stud Locat Anal 8 (1996), 49–58.

[26] R. Whitaker, A fast algorithm for the greedy interchange for large-scale clustering and median location problems, INFOR 21 (1983), 95–108.