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 $V = \{v_1, v_2, \ldots, v_p\}$ 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

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

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

$$f(X) = \max_{u_i \in U} \min_{v_j \in X} d'(u_i, v_j)$$  \hspace{1cm} (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]):

$$\text{Minimize } z$$  \hspace{1cm} (3)

subject to

---

Received August 2000; accepted April 2003

Correspondence to: P. Hansen

© 2003 Wiley Periodicals, Inc.
\[
\sum_{j} x_{ij} = 1, \quad \forall \ i,
\]
(4)
\[
x_{ij} \leq y_j, \quad \forall \ i, \ j,
\]
(5)
\[
\sum_{j} y_j = p,
\]
(6)
\[
z \geq \sum_{j} d_{ij} x_{ij}, \quad \forall \ i,
\]
(7)
\[
x_{ij}, \ y_j \in \{0, 1\}, \quad \forall \ i, \ j,
\]
(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 \( V \setminus S \) is adjacent to a vertex in \( S \). The problem is to find a dominating set \( S \) with minimum cardinality. Given a solution \( X = \{ v_{j1}, \ldots, v_{jk} \} \) for the \( p \)-Center problem, there obviously exists an edge \((u_i, v_{js})\) such that \( d(u_i, v_{js}) = f(X) \). 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 \( G_j \) 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 \(\ell^*\) 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
  \[g(\ell) = \begin{cases} 
  \max\{f, z(x_{cur}(\ell)), g_1\} & \text{if } \ell \neq \ell^*, \\
  \max\{f, z(x_{cur}(\ell)), g_2\} & \text{if } \ell = \ell^*. 
  \end{cases}\]

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

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 \( v_{new} \). 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 \( O(n) \).

**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,
\[ X_j = X \setminus \{v_j\} = X \cup \{v_{in}\} \setminus \{v_j\}, \quad v_j \in X. \]

Let us further denote new assignments of user \( u_i, i = 1, \ldots, m \) (in each solution \( X_j \)), by \( c'_j(i, j) \). 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, \( U' = \{u_i\} \)

---

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

**FIG. 3.** Example with \( m = n = 18 \) and \( p = 3 \); facility \( in = 6 \) is added to the solution.
It is easy to see that in addition to relations (9), (10), and (11): \( f = \max_{i \in U} \{d(i, in)\}. \) \hspace{1cm} (9)

In Figures 2 and 3, we see that \( v_m = v_o, U' = \{u_b, u_0\}, \) deleted edges are \((v_b, v_o)\) and \((v_b, v_1)\), and \( f = d(9, 6). \)

(ii) \( U' \) users that did not change their center \( j: U_j = S_j \cup U' \); that is, \( c'(i, j) = c(i) \). The radius of each group of users \( S_j \) is simply \( r(j) = \max_{i \in U_j} \{d(i, j)\} = c(i) \in X\}; \hspace{1cm} (10) \)

(iii) \( U' \) users that lost their center \( v_j \), but different from those in \( U' \): \( U_j = U \setminus (U_j \cup U') \). The new center for each such user \( u_i \) is its second closest [with index \( c_2(i) \)] or the new added one \( v_m \): \( c'(i, j) = \arg \min_{j \in U'} \{d(i, j), d(i, c_2(i))\} \Rightarrow j = c_1(i) \). Then, the radius is: \( r(j) = \max_{i \in U_j} \{d(i, c_1(i), j)\} \)

\[ = \max \{\min_{i \in U_j} \{d(i, j), d(i, c_2(i))\} : j = c_1(i) \} \}

\hspace{1cm} (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 \( U_j \) as well as relations (9), (10), and (11):

\[ f(X^{(best)}) = \min_{i \in X} \{f(X_i) = \min_{j \in X} \{\max_{i \in U} \{d(i, c_1(i), j)\}\}\} \]

\[ = \min_{i \in U} \{\max_{i \in U} \{d(i, c_1(i), j)\}, \max_{i \in U} \{d(i, c_2(i), j)\}\} \]

\[ \times \max_{i \in U_j} \{d(i, c_1(i), j)\} \}

\[ = \min_{i \in X} \{\max_{i \in U} \{f', z(j), \max_{i \in U_j} \{d(i, c_1(i), j)\}\} \}

\[ = \min_{i \in X} \{\max_{i \in U} \{f', z(j), \max_{i \in U_j} \{d(i, c_1(i), j)\}\} \}

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 \( N(X) \) 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 \( v_m \) 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.

**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 incoming 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 \( N(X) \) 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

1. \((\ast\) For users whose center is deleted, find new one \(\ast\)\)
   1. if \(c1(i) = goin\) then
      1. if \(d(i, goin) \leq d(i, c2(i))\) then
         1. \(c1(i) = goin\)
      2. else
         1. \(c1(i) = c2(i)\)
      2. \((\ast\) Find second closest facility for user \(i\) \(\ast)\)
         1. find center \(\ell^*\) where \(d(i, \ell)\) is minimum (for \(\ell = 1, \ldots, p, \ell \neq c1(i)\));
         2. \(c2(i) \leftarrow \ell^*\)
      3. endif
   2. else
      1. if \(d(i, c1(i)) > d(i, goin)\) then
         1. \(c2(i) \leftarrow c1(i)\) and \(c1(i) \leftarrow goin\)
      2. else
         1. if \(d(i, goin) < d(i, c2(i))\) then
            1. \(c2(i) = goin\)
         2. else
            1. if \(c2(i) = goout\) then
               1. \(\text{find center } \ell^*\text{ where } d(i, \ell)\text{ is minimum (for }\ell = 1, \ldots, p, \ell \neq c1(i)\));
               2. \(c2(i) \leftarrow \ell^*\)
            2. endif
         3. endif
      4. endif
   5. endif

5. end for \(i\)

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig5}
\caption{Pseudocode for procedure update within fast 1-interchange method.}
\end{figure}

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 \(j^* = c1(i^*)\) \(f = d(u_{i^*}, v_{j^*})\). We call \textit{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 \(S(X)\) if the facility to be added is farther from the critical user (with index \(i^*\)) than facility \(j^* = c1(i^*)\) (see Fig. 6). In other words, we have the following:

\textbf{Property 3}. \textit{If there is a better solution in the vertex substitution neighborhood \(S(X)\) 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 \(v_{j^*}\).}

\textbf{Proof}. Let \(i'\) and \(j' = c1(i')\) 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

\begin{equation}
J(i^*) = \{j | d(i^*, j) < f(X)\}.
\end{equation}

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 \textit{Move} and \textit{Update}. 

\textbf{FIG. 5.} Pseudocode for procedure update within fast 1-interchange method.
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 1 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 \((V'_{in}, V''_{in}, V_{out}, V'_{out})\). 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 \( V_{in} \) and replaced with one that belongs to the solution as well (i.e., from \( V''_{in} \), but whose turn has come to change Tabu status; in its place comes a facility from \( V'_{out} \), which is substituted by an element from \( V''_{out} \), and, finally, the chain is closed by placing the facility from \( V''_{in} \) in \( V''_{out} \) (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 \( V'_{in} \) and \( V''_{out} \) 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 \((V_m = \emptyset)\); (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 \(\chi = \{X|X = \text{set of } p \text{ (out of } m) \text{ locations of facilities}\}\) 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\( (in, out, cn_2, x_{c_{ur}}) \) | Chain – substitution – 2\( (in, out, cn_1, cn_2, x_{c_{ur}}) \)

\[
\begin{align*}
& a \leftarrow x_{c_{ur}}(out); \\
& x_{c_{ur}}(out) \leftarrow x_{c_{ur}}(cn_2); \\
& x_{c_{ur}}(cn_2) \leftarrow x_{c_{ur}}(in); \\
& x_{c_{ur}}(in) \leftarrow a;
\end{align*}
\]

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 \( f_{cur}, x_{cur}, c_1_{cur}, c_2_{cur} \) and \( i^*_{cur} \) respectively; (3) initialize Tabu lists \( (V_{in}^i = V_{out}^i = \emptyset) \) 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 \( J = \{ j \mid d(i^*_{cur}, j) < f_{cur} \} \setminus V_{out}^i \);

if \( J = \emptyset \) or \( improve = .false. \), set \( J = \{ x_{cur}(i) \mid i = p + 1, n \} \setminus V_{out}^i \).

(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

\( Move(c_1_{cur}, c_2_{cur}, d, x_{cur}, in, m, p, f_{cur}, out) \); (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

(2) Improvement.

if \( f^* < f_{opt} \) then

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

else set \( improve = .false. \).

(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:

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

locations. Since $\chi$ is a set of sets, a (symmetric) distance function $\rho$ can be defined as
\[
\rho(X_1, X_2) = |X_1 \setminus X_2| = |X_2 \setminus X_1|, \quad \forall X_1, X_2 \in X. \quad (13)
\]
It can easily be checked that $\rho$ is a metric function in $\chi$; thus, $\chi$ is a metric space. The neighborhood structures that we use are induced by the metric $\rho$, that is, $k$ locations of facilities ($k \leq p$) from the current solution are replaced by $k$ others. We denote by $N_k(X)$ the set of such neighborhood structures and by $N_k^p(X)$ the set of solutions forming neighborhood $N_k^p$ of a current solution $X$. More formally,
\[
X' \in N_k^p(X) \iff \rho(X', X) = k. \quad (14)
\]

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 with equal value; 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 $k_{\text{max}}$ (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: $|V_{\text{in}}^t| = t_1 = 1 + (p - 1) \cdot \text{Rnd}$ and $|V_{\text{out}}^t| = t_2 = 1 + (n - p - 1) \cdot \text{Rnd}$, where $\text{Rnd}$ 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 $[0, 100] \times [0, 100]$ 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 \( N_k (k = 1, \ldots, k_{max}) \) is induced by the distance function \( \rho \) (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 \( f_{cur}, x_{cur}, c_1_{cur}, c_2_{cur} \) 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 \( c_1_{cur} \) and \( c_2_{cur} \) for such interchange, i.e., run subroutine

\[ \text{Update} \ (d, \text{in}, \text{out}, \text{m}, \text{p}, c_1_{cur}, c_2_{cur}); \]
. Update \( x_{cur} \), \( f_{cur} \) and \( i^*_{cur} \) accordingly;

End For

(2b) Local search.

Apply algorithm \( I \) (without Initialization step), with \( x_{cur}, c_1_{cur}, c_2_{cur} \) 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 \( N_1 \) ∗)

\[ f_{opt} \leftarrow f_{cur}; \ {x}_{opt} \leftarrow {x}_{cur}; \ c_1 \leftarrow c_1_{cur}; \ c_2 \leftarrow c_2_{cur}; \ i^* \leftarrow i^*_{cur} \ \text{and set} \ k \leftarrow 1 \]

else

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

\[ f_{cur} \leftarrow f_{opt}; \ {x}_{cur} \leftarrow {x}_{opt}; \ c_1_{cur} \leftarrow c_1; \ c_2_{cur} \leftarrow c_2; \ i^*_{cur} \leftarrow i^* \ \text{and set} \ k \leftarrow k + 1 \]

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

\[ \frac{f_{ex} - f_{yp}}{f_{ex}} \times 100\%. \quad (15) \]

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.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( p )</th>
<th>( f_{ex} )</th>
<th>( f_{hp} )</th>
<th>% Gap</th>
<th>Time</th>
<th># Iter.</th>
<th># Nodes</th>
<th>M-I</th>
<th>VNS</th>
<th>TS</th>
<th>B-S</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>5</td>
<td>28.30</td>
<td>16.98</td>
<td>40.00</td>
<td>88.68</td>
<td>60,357</td>
<td>2652</td>
<td>28.30</td>
<td>28.30</td>
<td>28.30</td>
<td>41.16</td>
</tr>
<tr>
<td>10</td>
<td>20.04</td>
<td>8.39</td>
<td>58.13</td>
<td>446.89</td>
<td>388,483</td>
<td>42,664</td>
<td></td>
<td>20.04</td>
<td>20.04</td>
<td>20.04</td>
<td>21.88</td>
</tr>
<tr>
<td>40</td>
<td>5</td>
<td>30.10</td>
<td>17.69</td>
<td>41.23</td>
<td>1111.38</td>
<td>494,122</td>
<td>15,060</td>
<td>30.10</td>
<td>30.10</td>
<td>30.10</td>
<td>54.85</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20.04</td>
<td>9.70</td>
<td>51.60</td>
<td>28,517.64</td>
<td>18,061,725</td>
<td>1,247,390</td>
<td>20.04</td>
<td>20.04</td>
<td>20.04</td>
<td>28.30</td>
</tr>
</tbody>
</table>

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.

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

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.

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 2\( n \) 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;
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 $t_{\text{max}}$. For the results presented in Table 4, $t_{\text{max}} = 2n$ 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 each test, the same initial solution for TS-1, TS-2, and VNS is generated: $p$ facilities are chosen at random, fol-

---

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

<table>
<thead>
<tr>
<th>Pr. no.</th>
<th>$n$</th>
<th>$p$</th>
<th>Best known</th>
<th>Objective values</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-S</td>
<td>I</td>
<td>Gr</td>
<td>Gr + I</td>
<td>GrP</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>5</td>
<td>127</td>
<td>184</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>10</td>
<td>98</td>
<td>142</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>10</td>
<td>93</td>
<td>137</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>20</td>
<td>74</td>
<td>109</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>33</td>
<td>48</td>
<td>76</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>5</td>
<td>84</td>
<td>113</td>
</tr>
<tr>
<td>7</td>
<td>200</td>
<td>10</td>
<td>64</td>
<td>87</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>20</td>
<td>55</td>
<td>76</td>
</tr>
<tr>
<td>9</td>
<td>200</td>
<td>40</td>
<td>37</td>
<td>59</td>
</tr>
<tr>
<td>10</td>
<td>200</td>
<td>67</td>
<td>20</td>
<td>33</td>
</tr>
<tr>
<td>11</td>
<td>300</td>
<td>5</td>
<td>59</td>
<td>71</td>
</tr>
<tr>
<td>12</td>
<td>300</td>
<td>10</td>
<td>51</td>
<td>78</td>
</tr>
<tr>
<td>13</td>
<td>300</td>
<td>30</td>
<td>36</td>
<td>58</td>
</tr>
<tr>
<td>14</td>
<td>300</td>
<td>60</td>
<td>26</td>
<td>43</td>
</tr>
<tr>
<td>15</td>
<td>300</td>
<td>100</td>
<td>18</td>
<td>30</td>
</tr>
<tr>
<td>16</td>
<td>400</td>
<td>5</td>
<td>47</td>
<td>55</td>
</tr>
<tr>
<td>17</td>
<td>400</td>
<td>10</td>
<td>39</td>
<td>55</td>
</tr>
<tr>
<td>18</td>
<td>400</td>
<td>40</td>
<td>28</td>
<td>44</td>
</tr>
<tr>
<td>19</td>
<td>400</td>
<td>80</td>
<td>19</td>
<td>31</td>
</tr>
<tr>
<td>20</td>
<td>400</td>
<td>133</td>
<td>14</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>500</td>
<td>5</td>
<td>40</td>
<td>52</td>
</tr>
<tr>
<td>22</td>
<td>500</td>
<td>10</td>
<td>38</td>
<td>56</td>
</tr>
<tr>
<td>23</td>
<td>500</td>
<td>50</td>
<td>23</td>
<td>34</td>
</tr>
<tr>
<td>24</td>
<td>500</td>
<td>100</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>25</td>
<td>500</td>
<td>167</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>26</td>
<td>600</td>
<td>5</td>
<td>38</td>
<td>50</td>
</tr>
<tr>
<td>27</td>
<td>600</td>
<td>10</td>
<td>32</td>
<td>43</td>
</tr>
<tr>
<td>28</td>
<td>600</td>
<td>60</td>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>600</td>
<td>120</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>30</td>
<td>600</td>
<td>200</td>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>31</td>
<td>700</td>
<td>5</td>
<td>30</td>
<td>38</td>
</tr>
<tr>
<td>32</td>
<td>700</td>
<td>10</td>
<td>29</td>
<td>44</td>
</tr>
<tr>
<td>33</td>
<td>700</td>
<td>70</td>
<td>16</td>
<td>25</td>
</tr>
<tr>
<td>34</td>
<td>700</td>
<td>140</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>35</td>
<td>800</td>
<td>5</td>
<td>30</td>
<td>37</td>
</tr>
<tr>
<td>36</td>
<td>800</td>
<td>10</td>
<td>27</td>
<td>43</td>
</tr>
<tr>
<td>37</td>
<td>800</td>
<td>80</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td>38</td>
<td>900</td>
<td>5</td>
<td>29</td>
<td>38</td>
</tr>
<tr>
<td>39</td>
<td>900</td>
<td>10</td>
<td>23</td>
<td>37</td>
</tr>
<tr>
<td>40</td>
<td>900</td>
<td>90</td>
<td>14</td>
<td>21</td>
</tr>
</tbody>
</table>

Average time: 43.9, 0.1, 3.6, 3.8
Average % dev.: 0.0, 48.5, 62.4, 119.7, 90.1
No. solved: 40, 0, 0, 0, 1, 0, 0, 0, 15

60 NETWORKS—2003
owed 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.
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 $t_{\text{max}}$ are reported: From $t_{\text{max}} = n/10,000$ seconds per problem to $t_{\text{max}} = 2n$ seconds. (Each line of Table 5 gives average results, as the last line in Table 4, but for the different $t_{\text{max}}$). 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 $t_{\text{max}} = n/10,000$ (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>
<thead>
<tr>
<th>$t_{\text{max}}$ (seconds)</th>
<th>M-I</th>
<th>VNS</th>
<th>TS-1</th>
<th>TS-2</th>
<th>M-I</th>
<th>VNS</th>
<th>TS-1</th>
<th>TS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n/10,000$</td>
<td>55.55</td>
<td>49.05</td>
<td>37.11</td>
<td>66.36</td>
<td>0.08</td>
<td>0.07</td>
<td>0.03</td>
<td>0.02</td>
</tr>
<tr>
<td>$n/5000$</td>
<td>51.44</td>
<td>44.22</td>
<td>31.67</td>
<td>57.93</td>
<td>0.11</td>
<td>0.09</td>
<td>0.05</td>
<td>0.04</td>
</tr>
<tr>
<td>$n/1000$</td>
<td>41.84</td>
<td>30.44</td>
<td>25.98</td>
<td>22.43</td>
<td>0.22</td>
<td>0.28</td>
<td>0.17</td>
<td>0.28</td>
</tr>
<tr>
<td>$n/500$</td>
<td>36.34</td>
<td>23.26</td>
<td>24.51</td>
<td>16.27</td>
<td>0.49</td>
<td>0.58</td>
<td>0.24</td>
<td>0.46</td>
</tr>
<tr>
<td>$n/100$</td>
<td>32.41</td>
<td>12.16</td>
<td>20.57</td>
<td>7.69</td>
<td>1.67</td>
<td>2.26</td>
<td>0.88</td>
<td>4.41</td>
</tr>
<tr>
<td>$n/50$</td>
<td>28.94</td>
<td>8.61</td>
<td>19.65</td>
<td>7.00</td>
<td>4.10</td>
<td>4.51</td>
<td>1.57</td>
<td>2.46</td>
</tr>
<tr>
<td>$n/20$</td>
<td>27.14</td>
<td>4.55</td>
<td>17.98</td>
<td>3.37</td>
<td>8.05</td>
<td>7.52</td>
<td>3.21</td>
<td>5.83</td>
</tr>
<tr>
<td>$n/10$</td>
<td>23.86</td>
<td>2.00</td>
<td>13.59</td>
<td>0.11</td>
<td>163.56</td>
<td>133.67</td>
<td>91.30</td>
<td>33.24</td>
</tr>
<tr>
<td>$n/20$</td>
<td>24.19</td>
<td>0.70</td>
<td>13.37</td>
<td>0.18</td>
<td>24.19</td>
<td>0.70</td>
<td>13.37</td>
<td>0.18</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>$p$</th>
<th>Best known</th>
<th>M-I</th>
<th>VNS</th>
<th>TS-1</th>
<th>TS-2</th>
<th>M-I</th>
<th>VNS</th>
<th>TS-1</th>
<th>TS-2</th>
<th>M-I</th>
<th>VNS</th>
<th>TS-1</th>
<th>TS-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>307.48</td>
<td>329.66</td>
<td>317.00</td>
<td>307.48</td>
<td>307.88</td>
<td>7.21</td>
<td>3.10</td>
<td>0.00</td>
<td>0.13</td>
<td>762.15</td>
<td>578.81</td>
<td>981.91</td>
<td>315.24</td>
</tr>
<tr>
<td>100</td>
<td>215.67</td>
<td>234.10</td>
<td>220.06</td>
<td>219.59</td>
<td>215.67</td>
<td>8.55</td>
<td>2.04</td>
<td>1.82</td>
<td>0.00</td>
<td>994.28</td>
<td>570.60</td>
<td>587.98</td>
<td>683.49</td>
</tr>
<tr>
<td>150</td>
<td>174.83</td>
<td>195.21</td>
<td>174.83</td>
<td>174.83</td>
<td>177.23</td>
<td>11.66</td>
<td>0.00</td>
<td>0.00</td>
<td>1.37</td>
<td>602.95</td>
<td>52.99</td>
<td>984.27</td>
<td>895.88</td>
</tr>
<tr>
<td>200</td>
<td>157.00</td>
<td>181.18</td>
<td>157.88</td>
<td>157.00</td>
<td>157.09</td>
<td>15.40</td>
<td>0.56</td>
<td>0.00</td>
<td>0.06</td>
<td>238.56</td>
<td>747.00</td>
<td>661.67</td>
<td>903.46</td>
</tr>
<tr>
<td>250</td>
<td>137.54</td>
<td>165.47</td>
<td>140.98</td>
<td>137.54</td>
<td>138.92</td>
<td>20.31</td>
<td>2.50</td>
<td>1.00</td>
<td>0.00</td>
<td>640.69</td>
<td>103.72</td>
<td>952.57</td>
<td>998.18</td>
</tr>
<tr>
<td>300</td>
<td>123.33</td>
<td>158.38</td>
<td>123.33</td>
<td>124.60</td>
<td>123.44</td>
<td>28.42</td>
<td>0.00</td>
<td>0.46</td>
<td>0.62</td>
<td>354.46</td>
<td>264.59</td>
<td>783.24</td>
<td>992.13</td>
</tr>
<tr>
<td>350</td>
<td>118.02</td>
<td>146.82</td>
<td>118.02</td>
<td>118.32</td>
<td>118.07</td>
<td>33.08</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>351.76</td>
<td>937.70</td>
<td>732.09</td>
<td>791.76</td>
</tr>
<tr>
<td>400</td>
<td>107.65</td>
<td>137.57</td>
<td>107.65</td>
<td>114.11</td>
<td>113.22</td>
<td>27.79</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>723.01</td>
<td>904.88</td>
<td>892.71</td>
<td>893.49</td>
</tr>
<tr>
<td>450</td>
<td>101.51</td>
<td>135.09</td>
<td>101.51</td>
<td>101.98</td>
<td>101.14</td>
<td>33.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>627.58</td>
<td>674.55</td>
<td>973.90</td>
<td>757.54</td>
</tr>
<tr>
<td>500</td>
<td>94.37</td>
<td>124.33</td>
<td>94.37</td>
<td>97.20</td>
<td>96.14</td>
<td>31.75</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>351.76</td>
<td>937.70</td>
<td>732.09</td>
<td>791.76</td>
</tr>
<tr>
<td>Average</td>
<td>20.86</td>
<td>0.82</td>
<td>1.30</td>
<td>1.04</td>
<td>592.05</td>
<td>562.18</td>
<td>806.89</td>
<td>806.74</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 imple-

mentation 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 smaller $p$.

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


