Solving the uncapacitated facility location problem using tabu search

Minghe Sun*

Department of Management Science and Statistics, College of Business, The University of Texas at San Antonio, San Antonio, TX 78249, USA

Available online 1 September 2005

Abstract

A tabu search heuristic procedure is developed to solve the uncapacitated facility location problem. Tabu search is used to guide the solution process when evolving from one solution to another. A move is defined to be the opening or closing of a facility. The net cost change resulting from a candidate move is used to measure the attractiveness of the move. After a move is made, the net cost change of a candidate move is updated from its old value. Updating, rather than re-computing, the net cost changes substantially reduces computation time needed to solve a problem when the problem is not excessively large. Searching only a small subset of the feasible solutions that contains the optimal solution, the procedure is computationally very efficient. A computational experiment is conducted to test the performance of the procedure and computational results are reported. The procedure can easily find optimal or near optimal solutions for benchmark test problems from the literature. For randomly generated test problems, this tabu search procedure not only obtained solutions completely dominating those obtained with other heuristic methods recently published in the literature but also used substantially less computation time. Therefore, this tabu search procedure has advantage over other heuristic methods in both solution quality and computation speed.

Keywords: Facility location; Tabu search; Heuristics; Optimization

* Tel.: +1 210 458 5777; fax: +1 210 458 6350.
E-mail address: minghe.sun@utsa.edu
URL: http://faculty.business.utsa.edu/msun.
The success or failure of business and public facilities depends in a large part on their locations. Effective supply chain management has led to increased profit, increased market share, reduced operating cost, and improved customer satisfaction for many businesses. One strategic decision in supply chain management is facility location [1]. Due to their strategic nature, facility location problems have been widely studied by researchers and practitioners over many years [2–4]. There are a variety of models representing a variety of facility location problems [3,5–9]. Most of these problems are combinatorial in nature. Therefore, exact algorithms exist only for small problems and heuristic procedures have to be used for large practical problems. In this study, a tabu search (TS) heuristic procedure is developed to solve the uncapacitated facility location (UFL) problem. In a UFL problem, a fixed cost is associated with the establishment of each facility and a fixed cost is associated with the opening and using of each road from a customer to a facility. The objective of a UFL problem is to decide where to locate the facilities and which roads to use so as to minimize the total cost.

The TS metaheuristic [10–14] is used in the procedure to guide the search process when moving from one solution to another in order to find good solutions. The procedure is computationally very efficient because the solution space being searched is only a small subset of all feasible solutions that contains the optimal solution. In the procedure, a move is the opening or closing of a facility. Net cost changes resulting from candidate moves are used to measure the attractiveness of the moves. Net cost changes are updated from their old values after a move is made. Steps are developed to update net cost changes and theoretical foundation supporting these steps is developed. Updating, rather than re-computing, net cost changes substantially reduces the computation time needed to solve a problem as long as the problem is not excessively large. The performance of the procedure is tested through computational experiments using test problems both from the literature and randomly generated. Computational results show that the TS procedure can easily find optimal or near optimal solutions for all test problems with known optimal solutions. With randomly generated test problems, the TS procedure found solutions better than those found by competitive heuristic methods recently published in the literature.

The rest of this paper is organized as follows. The UFL problem is briefly discussed in Section 1. The TS heuristic procedure is developed in Section 2. The computation and update of net cost changes are described in Section 3. An example demonstrating the computation and update of net cost changes is presented in Section 4. Computational results are reported in Section 5. Conclusions and further remarks are given in Section 6.

1. The uncapacitated facility location problem

A UFL problem with m customers and n candidate facility sites can be represented by a network with m + n nodes and mn arcs [3,15]. In the UFL model, \( f_j \) is used to represent the cost of opening facility \( j \) and \( c_{ij} \) is used to represent the cost of serving customer \( i \) from facility \( j \) or assigning customer \( i \) to facility \( j \). We assume that \( c_{ij} \geq 0 \) for all \( i = 1, \ldots, m \) and \( j = 1, \ldots, n \) and \( f_j > 0 \) for all \( j = 1, \ldots, n \). A binary variable \( y_j \) is used to represent the status of facility \( j \) in the model. Facility \( j \) will be open only if \( y_j = 1 \) in the solution. A binary variable \( x_{ij} \) is used for the road from customer \( i \) to facility \( j \) in the model. Customer \( i \) will be served by facility \( j \) only if \( x_{ij} = 1 \) in the solution. However, each \( x_{ij} \) can be treated as a continuous variable and will have a binary value in the solution [15]. The solution process of the UFL problem is to find an optimal solution that satisfies all customer demand and minimizes the
total cost (1). The UFL problem can be formally stated as [15]

\[
\min \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij}x_{ij} + \sum_{j=1}^{n} f_j y_j
\]

\[\text{s.t.} \sum_{j=1}^{n} x_{ij} = 1 \quad \text{for} \quad i = 1, \ldots, m, \quad (2)\]

\[x_{ij} \leq y_j \quad \text{for} \quad i = 1, \ldots, m \quad \text{and} \quad j = 1, \ldots, n, \quad (3)\]

\[x_{ij} \geq 0 \quad \text{for} \quad i = 1, \ldots, m \quad \text{and} \quad j = 1, \ldots, n, \quad (4)\]

\[y_j = 0 \text{ or } 1 \quad \text{for} \quad j = 1, \ldots, n. \quad (5)\]

In the literature, the UFL problem is also called the simple facility location problem, the simple (or uncapacitated) warehouse location problem or the simple (or uncapacitated) plant location problem. Many successful applications of the UFL model have been reported. Many practical problems without facilities to locate, such as cluster analysis, machine scheduling, economic lot sizing, portfolio management [15], and computer network design [16], can also be modeled as UFL problems.

Developing solution methods for the UFL problem has been a hot topic of research for the last 40 years. Kuehn and Hamburger [17] developed the first heuristic that has two phases. The first phase is a greedy approach, called the ADD method, that starts with all facilities closed, keeps adding (opening) the facility resulting in the maximum decrease in the total cost (1), and stops if adding any more facility will no longer reduce the total cost. The second phase is a local search method in which an open facility and a closed facility are interchanged as long as such an interchange reduces the total cost. Another greedy heuristic is the DROP method that starts with all facilities open, keeps dropping (closing) the facility that gives the maximum decrease in the total cost, and stops if dropping any more facility will no longer reduce the total cost [3,18,19]. These early heuristics provided the basis for many sophisticated heuristics and provided an initial incumbent for many exact solution algorithms [18,19]. Erlenkotter [20] developed a dual approach for the UFL problem. Although this dual approach is an exact algorithm, it can also be used as a heuristic to find good solutions. One effective and widely used heuristic is the Lagrangian method [21] that is based on Lagrangian relaxation and subgradient optimization [3]. More recently Gen et al. [16] and Vaithyanathan et al. [22] used artificial neural network approaches to solve UFL problems. Although different researchers [23–28] have applied TS to the UFL problem with different degrees of success, there are differences in implementation details and there is plenty room for further improvement. Furthermore, procedures developed for the closely related \(p\)-median problem, such as the one reported by Rolland et al. [29], can be modified to solve the UFL problem. TS procedures have also been developed to solve more complicated facility location problems, such as the ones by Crainic and Gendreau [30], Delmaire et al. [31], Filho and Galvão [32], and Tuzun and Burke [33].

In addition to heuristics, there are a variety of exact algorithms for the UFL problem, such as the dual approach of Erlenkotter [20] and the primal-dual approaches of Körkel [34]. Because the UFL problem is NP-hard [15], exact algorithms may not be able to solve large practical problems. The UFL problem has been studied extensively and many researchers have made great contributions in developing exact and heuristic solution methods. Krarup and Pruzan [8] and Cornuéjols et al. [15] gave excellent surveys and reviews of applications and solution methods. However, reviewing these contributions is not the purpose of this study and only publications closely related to this study are cited.
Some UFL test problems used in previous studies and reported in the literature are collected, stored and updated by different researchers, such as Beasley [35] and Hoefer [36]. Therefore, researchers can use the same sets of test problems to test their algorithms or heuristic procedures. Hence, the performances of different heuristic procedures, especially their effectiveness in finding good solutions, become comparable even if they are not directly compared through a computational experiment.

2. The TS heuristic procedure

TS [10–14] is a metaheuristic method for solving combinatorial optimization problems. It uses flexible memory and responsive exploration in guiding the solution process to move from one trial solution to another. By flexible memory, it has three memory structures, called short term, medium term and long term memory processes. By responsive exploration, it determines a search direction in the solution space based on the properties of the current solution and the search history. We start with a discussion of the solution space being searched and then describe the components of the heuristic procedure. The solution process consists of different search cycles. Each search cycle, except for the first one, consists of the long term memory process, the short term memory process, and the medium term memory process. A step-by-step outline of the TS procedure is presented after the components are discussed.

2.1. The solution space

Let \( J \) denote the index set of all candidate facilities. Partition \( J \) into two subsets \( J_0 \) and \( J_1 \), where \( J_0 \) consists of the indices of closed facilities and \( J_1 \) consists of the indices of open facilities, i.e.,

\[
J_0 = \{ j | y_j = 0 \} \quad \text{and} \quad J_1 = \{ j | y_j = 1 \}.
\]

(6)

We use \( n_1 \) to denote the number of open facilities of a given partition, i.e., \( n_1 = |J_1| \).

For each customer \( i \), \( d^1_i \) is defined to be the index of the facility, such that

\[
c_{id^1_i} = \min \{ c_{ij} | j \in J_1 \}.
\]

(7)

If more than one \( j \) satisfies (7), any one of them can be designated as \( d^1_i \). The solution obtained by assigning each customer \( i \) to the corresponding facility \( d^1_i \) is a feasible solution [15]. Because such a solution also has the lowest total cost (1) among all feasible solutions for the given partition of \( J \), we only need to evaluate one solution for each partition of \( J \) without the risk of missing the optimal solution. The total cost (1) of the solution corresponding to a partition of \( J \) can be computed using the following:

\[
z = \sum_{i=1}^{m} c_{id^1_i} + \sum_{j \in J_1} f_j.
\]

(8)

Solutions corresponding to the partitions of \( J \) constitute the solution space. Therefore, the solution process of a UFL problem is to find a partition of \( J \) with a corresponding solution that minimizes the total cost (8). The TS procedure uses the strategy of searching for the best partition of \( J \). Because only one solution that achieves the minimum total cost (8) is evaluated for each given partition of \( J \), the TS procedure is computationally very efficient.
When \( n_1 > 1 \), we also define \( d^2_i \) for each customer \( i \) to be the facility, such that

\[
c_{id^2_i} = \min\{c_{ij} | j \in J_1 \land j \neq d^1_i\}.
\]  

(9)

If facility \( d^1_i \) is closed, customer \( i \) will be reassigned to facility \( d^2_i \). When \( n_1 = 1 \), \( d^2_i \) is not defined and is not needed.

2.2. Move

The status of facility \( j \) in a given partition of \( J \) is given by the value of \( y_j \). A move is defined to be the status change of any facility \( j \), i.e., \( y_j \leftarrow 1 - y_j \). Thus a move leads from the current partition to a new partition of \( J \). The two solutions before and after a move are adjacent because one can be reached from the other within one move.

A solution corresponding to a partition is feasible if \( n_1 \geq 1 \). Changing the status of any facility is a candidate move as long as the resulting solution is feasible. Therefore, the only restriction on a candidate move is that facility \( j \) cannot be closed if it is the only open facility. At each feasible solution, there are \( n \) different candidate moves when \( n_1 > 1 \) and \( n - 1 \) candidate moves when \( n_1 = 1 \).

We use \( k \) to count the number of moves, or iterations, made since the start of the search process. We use \( \Delta z^k_j \) to represent the resulting net cost change of the move switching the status of facility \( j \) at iteration \( k \). The value of \( \Delta z^k_j \) is used to measure the attractiveness of the corresponding move. The lower the value of \( \Delta z^k_j \), the more attractive the move. The value of \( \Delta z^k_j \) is computed or updated for each \( j \in J \) before a move is selected. The details of the computation or update of the \( \Delta z^k_j \)'s are discussed in the next section.

Complicated moves, such as swap moves in which an open facility is closed and a closed facility is opened at the same time, used by other researchers (e.g., [25,33]) are not used in this study because the computation of the net cost change resulting from such more complicated moves are more time consuming. Nevertheless, a more complicated move can be decomposed into multiple simple moves.

2.3. The short term memory process

Recency based memory, represented by the integer vector \( t \in \mathbb{R}^n \), is used to implement the short term memory process. The element \( t_j \) represents the move number, i.e., the value of \( k \), at which facility \( j \) changed status the last time. Each time when facility \( j \) changes status, \( t_j \) is updated. The total cost of the best solution found in the current search cycle is denoted by \( z_0 \) that is updated each time a better solution is found. The move number when \( z_0 \) is updated is denoted by \( k_0 \). The total cost of the best solution found in the whole search process is denoted by \( z_{00} \) and \( z_{00} \) is updated any time a solution with a total cost less than \( z_{00} \) is found. We use the integer \( l_c \) (\( l_o \)) to represent the tabu size for those facilities that are currently closed (open), i.e., each facility is kept closed (open) for at least \( l_c \) (\( l_o \)) moves after it is closed (opened) unless the aspiration criterion is satisfied. A move switching the status of facility \( j \) is labeled tabu if facility \( j \) was closed within the last \( l_c \) moves or was opened within the last \( l_o \) moves. The value of \( l_c \) (\( l_o \)) is allowed to vary from one search cycle to the next between its lower limit \( l_{c1} \) (\( l_{o1} \)) and upper limit \( l_{c2} \) (\( l_{o2} \)), i.e., \( l_{c1} \leq l_c \leq l_{c2} \) (\( l_{o1} \leq l_o \leq l_{o2} \)); however, it is kept constant in the duration of a search cycle and is reset only when the short term memory process restarts in a new search cycle.
For each move, we select a facility $\bar{j}$ to switch status, such that
\[
\Delta z^k_j = \min\{\Delta z^k_j | j = 1, \ldots, n \text{ and facility } j \text{ is not flagged}\}.
\]

Then the following tabu condition is checked
\[
k - t_j \leq l_c \text{ if } y_j = 0 \text{ or } k - t_j \leq l_o \text{ if } y_j = 1.
\]

The move is tabu if (11) is satisfied. The selected move is made if it is not tabu; otherwise, the following aspiration criterion (12) is checked:
\[
z + \Delta z^k_j < z_0,
\]
i.e., if the resulting solution is better than the best solution found in the current search cycle. This aspiration criterion is similar to those used by Sun [37], Sun et al. [38] and Sun and McKeown [39]. If this aspiration criterion is satisfied, the move will be made even if it is tabu. Otherwise, if the selected move is tabu and this aspiration criterion is not satisfied, the move cannot be made and facility $\bar{j}$ is flagged. In this case, another facility is selected according to (10), the tabu status of the newly selected move is checked, and so on. This process is repeated until a move that is not tabu, or tabu but satisfies the aspiration criterion (12), is found and the move is made.

Let $\alpha_1 > 0$ be a coefficient such that the short term memory process will stop if no solution better than $z_0$ can be found after $\alpha_1 n$ moves, i.e., the short term memory process stops once the following condition is satisfied:
\[
k - k_0 > \alpha_1 n.
\]

2.4. The medium term memory process

In addition to the recency based memory, the frequency based memory, represented by the integer vector $h \in \mathbb{R}^n$, is also used to implement the medium term memory process. The element $h_j \in h$ represents the frequency, i.e., the number of moves, that $j$ is in $J_1$ since the search started. The value of $h_j$ is updated only when facility $j$ changes status from open to closed.

In the medium term memory process, we change the criterion used to select a facility $\bar{j}$ for a move. Instead of using criterion (10) in the short term memory process, we use the following criterion:
\[
\Delta' z^k_j = \min\{\Delta' z^k_j | j = 1, \ldots, n \text{ and facility } j \text{ is not flagged}\}.
\]
The $\Delta' z^k_j$'s are defined in the following:
\[
\Delta' z^k_j = \begin{cases} 
\Delta z^k_j - \frac{h_j}{n} n - n_1 \bar{f} & \text{if } y_j = 0, \\
\Delta z^k_j + \frac{h_j - h_j - l_j}{n} n - n_1 \bar{f} & \text{if } y_j = 1,
\end{cases}
\]
where $\bar{f}$ is the average of the $f_j$’s, i.e., $\bar{f} = (1/n) \sum_{j=1}^n f_j$. With the criterion in (15), a facility $j$ with $y_j = 0$ and a large $h_j$ is more likely to be selected to open; a facility $j$ with $y_j = 0$ and a small $h_j$ is more likely to stay closed; a facility $j$ with $y_j = 1$ and a large $h_j$ is more likely to stay open; and a facility with $y_j = 1$ and a small $h_j$ is more likely to be selected to close. As a result, facilities that are frequently closed are more likely to be closed and facilities that are frequently open are more likely to be open so as to achieve intensification.
Once a solution better than \( z_0 \) is found in the medium term memory process, the short term memory process automatically restarts. Let \( x_2 > 0 \) be a coefficient such that the medium term memory process will stop if no better solution can be found after \( x_2n \) moves, i.e., the medium term memory process stops if the following is satisfied:

\[
    k - k_0 > (x_1 + x_2)n.
\]

Once the medium term memory process stops, the procedure tries to search for a local optimal solution. A move is made to switch the status of facility \( j \) if \( \Delta z_j^k < 0 \). A local optimal solution is reached if \( \Delta z_j^k \geq 0 \) for all \( j \in J \).

2.5. The long term memory process

The recency based memory is used to implement the long term memory process. The procedure allows the long term memory process to be invoked \( C \) times and \( c \) is used to count the number of times it has been invoked. When it is invoked the \( c \)th time, \( c \) moves are made in the current long term memory process and \( c_0 \) is used to count the number of moves that has been made. In a move, a facility \( \bar{j} \) is chosen to switch status according to the following criterion:

\[
    t_j = \min\{t_j | j = 1, \ldots, n\}.
\]

Given the tabu condition in (11), facility \( \bar{j} \) is kept at its new status for at least \( l_c \) moves if it is closed or \( l_o \) moves if it is opened in the move. By doing so, the search is led to a new region in the solution space to achieve diversification. The strategy used in the long term memory process is similar to those used by Sun et al. [38] and by Sun and McKeown [39].

2.6. Stopping rule

At the end of each search cycle, the procedure checks if the long term memory process has been invoked \( C \) times. If not, the procedure invokes the long term memory process to start a new search cycle; otherwise, it terminates with the best solution found as the final solution. The short term memory process starts each time after the long term memory process finishes and the medium term memory process follows the short term memory process. One search cycle follows another until the long term memory process has been invoked \( C \) times.

We call \( l_1^c, l_2^c, l_0^1, l_0^2, x_1, x_2 \) and \( C \) the parameters of the TS procedure. Users choose values for these parameters before the solution process starts. By varying the values of these parameters, we can control the thoroughness and extensiveness of the search process.

2.7. The tabu search procedure

The search process starts from a local optimal solution that can be obtained with any greedy method. Although the DROP method was used in the implementation, no differences in performance were noticed when the ADD method was used. The following is a step-by-step description of the procedure.

\textbf{Step 0.} Find a local optimal solution with a greedy method. Let \( z \) represent the total cost of the current solution. Let \( z_0 \leftarrow z \) and \( z_{00} \leftarrow z \). Determine the value of \( n_1 \). Select values for \( l_1^c, l_2^c, l_0^1 \) and \( l_0^2 \) and
determine the initial tabu sizes $l_c$ and $l_o$ such that $l_1^c \leq l_c \leq l_2^c$ and $l_1^o \leq l_o \leq l_2^o$. Let $t_j \leftarrow -l_c$ for all $j \in J_0$ and $t_j \leftarrow -l_o$ for all $j \in J_1$ to initialize the vector $t$. Let $h_j \leftarrow 0$ for all $j \in J_0$ and $h_j \leftarrow t_j$ for all $j \in J_1$ to initialize the vector $h$. Select values for $z_1$, $z_2$ and $C$. Let $k \leftarrow 1$, $k_0 \leftarrow 1$, $c_0 \leftarrow 1$ and $c \leftarrow 1$. Compute $\Delta z_j^1$ with (19) if $y_j = 0$ or (21) if $y_j = 1$ and mark each facility $j$ as unflagged.

**Step 1.** Select a facility $\bar{j}$ according to (10). Check the tabu status of the selected move (11). If tabu, go to Step 2; otherwise, go to Step 3.

**Step 2.** Check the aspiration criterion of the selected move (12). If (12) is satisfied, go to Step 3; otherwise, mark facility $\bar{j}$ as flagged and go to Step 1.

**Step 3.** If $y_j = 1$, let $n_1 \leftarrow n_1 - 1$ and let $h_j \leftarrow h_j + k - t_j$; otherwise let $n_1 \leftarrow n_1 + 1$. Let $y_j \leftarrow 1 - y_j$, $z \leftarrow z + \Delta z_j^k$, $t_j \leftarrow k$ and $k \leftarrow k + 1$. If $z < z_0$, let $z_0 \leftarrow z$ and $k_0 \leftarrow k$. If $z < z_{00}$, let $z_{00} \leftarrow z$.

**Step 4.** Update $\Delta z_j^k$ using (24), (26), (28) or (31). Mark each facility $j$ as unflagged. If $k - k_0 \leq x_1 n$, go to Step 1; if $k - k_0 \leq (x_1 + x_2) n$, continue to Step 5; otherwise, go to Step 8.

**Step 5.** Compute $\Delta' z_j^k$ using (15).

**Step 6.** Select a facility $\bar{j}$ according to (14). Check the tabu status of the selected move according to (11). If not tabu, go to Step 3.

**Step 7.** Check the aspiration criterion of the selected move (12). If (12) is satisfied, go to Step 3; otherwise, mark facility $\bar{j}$ as flagged and go to Step 6.

**Step 8.** If a local optimal solution has not been found, select a facility $\bar{j}$ according to (10) and go to Step 3.

**Step 9.** If $c > C$, Stop. If $c_0 > c$, go to Step 11.

**Step 10.** Let $c_0 \leftarrow c_0 + 1$. Select a facility $\bar{j}$ according to (17) and go to Step 3.

**Step 11.** Let $c_0 \leftarrow 1$, $c \leftarrow c + 1$, $z_0 \leftarrow z$ and $k_0 \leftarrow k$. Reset the value of $l_c$ and $l_o$ such that $l_1^c \leq l_c \leq l_2^c$ and $l_1^o \leq l_o \leq l_2^o$ and go to Step 4.

These steps are used jointly to implement the three memory processes and some steps are shared by more than one memory process. However, a rough division may be made among them. Steps 1–4 are used to implement the short term memory process; Steps 3–8 are used to implement the medium term memory process; and Steps 3, 4, 9 and 10 are used to implement the long term memory process.

Different researchers used different strategies when developing their TS procedures. Al-Sultan and Al-Fawzan [23] employed only the short term memory process and randomly selected a subset of candidate moves to evaluate at each trial solution. Ghosh [25] used swap moves but found that a short term memory process, usually the main search process [10–12], did not help the performance of the procedure. Sun [28] recomputed the net cost changes after each move and did not use a medium term memory process. Emphasizing simplicity and robustness, Michel and Van Hentenryck [26] used only a simple short term memory process.

### 3. Computation and update of net cost changes

In this section, we first discuss how the $\Delta z_j^k$’s are computed at the current partition of $J$ and then discuss how they are updated after a move is made. For notational convenience, we use $z^k$ to denote the total cost (8) at move $k$. Whitaker [40] developed similar procedures for computing and updating the $\Delta z_j^k$’s for the median location problem. Michel and Van Hentenryck [26] also used similar strategies to update the $\Delta z_j^k$’s.
3.1. Computation of the net cost changes

Given the current partition of $J$, each customer $i$ is assigned to facility $d_i^1$ in the corresponding solution. If facility $j$, such that $j \in J_0$, is to open in the next move, customer $i$ will be reassigned to facility $j$ only if $c_{ij} < c_{id_i^1}$. Otherwise, customer $i$ will stay with facility $d_i^1$. Define the set $P_j$ as

$$P_j = \{i | c_{ij} < c_{id_i^1}\}.$$  (18)

**Theorem 1.** For each $j \in J_0$, the net cost change $\Delta z_j^k$ is given by

$$\Delta z_j^k = f_j + \sum_{i \in P_j} (c_{ij} - c_{id_i^1}).$$  (19)

**Proof.** Given the current partition of $J$ into $J_0$ and $J_1$, the total cost $z_j^{k-1}$ for each $j \in J_0$ (8) can be rewritten as

$$z_j^{k-1} = \sum_{i \notin P_j} c_{id_i^1} + \sum_{i \in P_j} c_{id_i^1} + \sum_{j' \in J_1} f_{j'}.$$  (20)

If facility $j$ is to open in the next move, the total cost $z_j^k$ for each $j \in J_0$ (8) can be rewritten as

$$z_j^k = \sum_{i \notin P_j} c_{id_i^1} + \sum_{i \in P_j} c_{ij} + \sum_{j' \in J_1} f_{j'} + f_j.$$  (21)

Therefore, the net cost change $\Delta z_j^k$ for each $j \in J_0$ is given by

$$\Delta z_j^k = z_j^k - z_j^{k-1} = f_j + \sum_{i \in P_j} (c_{ij} - c_{id_i^1}).$$  (22)

After facility $j$ opens in move $k$, the indices $d_i^1$ and $d_i^2$ for each $i \in P_j$ are updated to $d_i^2 = d_i^1$ and $d_i^1 = j$.

If $n_1 \geq 2$, a facility $j \in J_1$ may be closed and the resulting solution is still feasible. Each customer $i$ currently assigned to facility $j$, i.e., $d_i^1 = j$, will be reassigned to facility $d_i^2$ after facility $j$ closes. Define the set $Q_j$ as

$$Q_j = \{i | d_i^1 = j\}.$$  (23)

**Theorem 2.** When $n_1 \geq 2$ for each $j \in J_1$, the net cost change $\Delta z_j^k$ is given by

$$\Delta z_j^k = -f_j + \sum_{i \in Q_j} (c_{id_i^2} - c_{ij}).$$  (24)

**Proof.** Given the current partition of $J$ into $J_0$ and $J_1$, the total cost $z_j^{k-1}$ for each $j \in J_1$ (8) can be rewritten as

$$z_j^{k-1} = \sum_{i \notin Q_j} c_{id_i^1} + \sum_{i \in Q_j} c_{ij} + \sum_{j' \in J_1} f_{j'}.$$  (25)
If facility $j$ is to close in the next move, the total cost $z^k$ for each $j \in J_1$ (8) can be rewritten as

$$z^k = \sum_{i \notin Q_j} c_{id_i} + \sum_{i \in Q_j} c_{id_i} + \sum_{j' \in J_1} f_{j'} - f_j.$$ 

Therefore, $\Delta z^k_j$ for each $j \in J_1$ is given by

$$\Delta z^k_j = z^k - z^{k-1} = -f_j + \sum_{i \in Q_j} (c_{id_i} - c_{ij}).$$

After facility $j$ closes in move $k$, $d_1^i$ is updated to $d_1^i = d_2^i$ and $d_2^i$ is determined through (9) for each $i \in Q_j$.

3.2. Update of the net cost changes

After a move is made, the values of the $\Delta z^k_j$’s can also be updated from the value of $\Delta z^{k-1}_j$ rather than recomputed using (19) or (21). Whether facility $\bar{j}$ was opened or closed in move $k-1$, the value of $\Delta z^k_j$ is

$$\Delta z^k_j = -\Delta z^{k-1}_j.$$ 

(22)

The update of $\Delta z^k_j$ from $\Delta z^{k-1}_j$ is discussed in four mutually exclusive situations for each $j \neq \bar{j}$.

3.2.1. Closed facilities after another facility opened

Suppose facility $\bar{j}$ was opened in move $k-1$. Define two sets $S^1_j$ and $S^2_j$ for each $j \in J_0$ as follows:

$$S^1_j = \{i \mid c_{ij} < c_{i\bar{j}} < c_{id_i} \} \quad \text{and} \quad S^2_j = \{i \mid c_{i\bar{j}} \leq c_{ij} < c_{id_i} \}.$$ 

(23)

Theorem 3. After facility $\bar{j}$ is opened in move $k-1$, $\Delta z^k_j$ is updated from $\Delta z^{k-1}_j$ for each $j \in J_0$ according to

$$\Delta z^k_j = \Delta z^{k-1}_j + \sum_{i \in S^1_j} (c_{id_i} - c_{ij}) + \sum_{i \in S^2_j} (c_{id_i} - c_{ij}).$$ 

(24)

Proof. Before facility $\bar{j}$ is opened in move $k-1$, $\Delta z^{k-1}_j$ for each $j \in J_0$ (19) can be rewritten as

$$\Delta z^{k-1}_j = f_j + \sum_{i \in (P_j - S^1_j - S^2_j)} (c_{ij} - c_{id_i}) + \sum_{i \in S^1_j} (c_{ij} - c_{id_i}) + \sum_{i \in S^2_j} (c_{ij} - c_{id_i}).$$

After facility $\bar{j}$ is opened in move $k-1$, $\Delta z^k_j$ for each $j \in J_0$ (19) can be rewritten as

$$\Delta z^k_j = f_j + \sum_{i \in (P_j - S^1_j - S^2_j)} (c_{ij} - c_{id_i}) + \sum_{i \in S^1_j} (c_{ij} - c_{ij}).$$
Therefore, we have
\[
\Delta z^k_j - \Delta z^{k-1}_j = \sum_{i \in S^1_j} (c_{ij} - c_{j\tilde{i}}) - \sum_{i \in S^2_j} (c_{ij} - c_{i\tilde{d}_i^1}) - \sum_{i \in S^2_j} (c_{ij} - c_{i\tilde{d}_i^2}) \\
= \sum_{i \in S^1_j} (c_{id_i^1} - c_{ij}) + \sum_{i \in S^2_j} (c_{id_i^2} - c_{ij}).
\]

For each \( i \in S^1_j \cup S^2_j \), \( d_i^2 \) is updated to \( d_i^2 = d_i^1 \) and \( d_i^1 \) is updated to \( d_i^1 = \tilde{j} \) after \( \Delta z^k_j \) is updated.

3.2.2. Open facilities after another facility opened

Suppose facility \( \tilde{j} \) was opened in move \( k - 1 \). Define two sets \( T^1_j \) and \( T^2_j \) for each \( j \in J_1 \) as follows:
\[
T^1_j = \{ i | d_i^1 = j \land c_{ij} < c_{ij} \} \quad \text{and} \quad T^2_j = \{ i | d_i^1 = j \land c_{ij} \leq c_{ij} < c_{id_i^2} \}.
\]  

**Theorem 4.** After facility \( \tilde{j} \) is opened in move \( k - 1 \), \( \Delta z^k_j \) is updated from \( \Delta z^{k-1}_j \) for each \( j \in J_1 \) according to
\[
\Delta z^k_j = \Delta z^{k-1}_j + \sum_{i \in T^1_j} (c_{ij} - c_{id_i^1}) + \sum_{i \in T^2_j} (c_{ij} - c_{id_i^2}).
\]  

**Proof.** Before facility \( \tilde{j} \) is opened in move \( k - 1 \), \( \Delta z^{k-1}_j \) for each \( j \in J_1 \) (21) can be rewritten as
\[
\Delta z^{k-1}_j = -f_j + \sum_{i \in (Q_j-T^1_j-T^2_j)} (c_{id_i^2} - c_{ij}) + \sum_{i \in T^1_j} (c_{id_i^1} - c_{ij}) + \sum_{i \in T^2_j} (c_{id_i^2} - c_{ij}).
\]

After facility \( \tilde{j} \) is opened in move \( k - 1 \), \( \Delta z^k_j \) for each \( j \in J_1 \) (21) can be rewritten as
\[
\Delta z^k_j = -f_j + \sum_{i \in (Q_j-T^1_j-T^2_j)} (c_{id_i^2} - c_{ij}) + \sum_{i \in T^2_j} (c_{id_i^2} - c_{ij}).
\]

As a result we have
\[
\Delta z^k_j - \Delta z^{k-1}_j = \sum_{i \in T^2_j} (c_{ij} - c_{id_i^1}) - \sum_{i \in T^1_j} (c_{id_i^1} - c_{ij}) - \sum_{i \in T^2_j} (c_{id_i^2} - c_{ij}) \\
= \sum_{i \in T^1_j} (c_{ij} - c_{id_i^1}) + \sum_{i \in T^2_j} (c_{ij} - c_{id_i^2}).
\]

For each \( i \in T^1_j \), \( d_i^2 \) is updated to \( d_i^2 = d_i^1 \) and \( d_i^1 \) is updated to \( d_i^1 = \tilde{j} \), and for each \( i \in T^2_j \), \( d_i^2 \) is updated to \( d_i^2 = \tilde{j} \), after \( \Delta z^k_j \) is updated. If facility \( j \) was the only one such that \( j \in J_1 \) before facility \( \tilde{j} \) was opened, then \( \Delta z^{k-1}_j \) was not defined and \( \Delta z^k_j \) cannot be updated with (26). Therefore, \( \Delta z^k_j \) is recomputed using (21) and \( d_i^1 \) and \( d_i^2 \) are determined through (7) and (9).
3.2.3. Closed facilities after another facility closed

Suppose facility $\bar{j}$ was closed in move $k - 1$. Define two sets $S^1_j$ and $S^2_j$ for $j \in J_0$ as follows:

$$S^1_j = \{ i | d^1_i = \bar{j} \land c_{ij} < c_{i\bar{j}} \} \quad \text{and} \quad S^2_j = \{ i | d^1_i = \bar{j} \land c_{ij} \leq c_{i\bar{j}} < c_{i\bar{d}}^2 \}. \quad (27)$$

**Theorem 5.** After facility $\bar{j}$ is closed in move $k - 1$, $\Delta z^k_j$ is updated from $\Delta z^{k-1}_j$ for each $j \in J_0$ according to

$$\Delta z^k_j = \Delta z^{k-1}_j + \sum_{i \in S^1_j} (c_{ij} - c_{i\bar{d}}^1) + \sum_{i \in S^2_j} (c_{ij} - c_{i\bar{j}}). \quad (28)$$

**Proof.** Before facility $\bar{j}$ was closed in move $k - 1$, $\Delta z^{k-1}_j$ for each $j \in J_0$ (19) can be rewritten as

$$\Delta z^{k-1}_j = f_j + \sum_{i \in (P_j - S^1_j)} (c_{ij} - c_{i\bar{d}}) + \sum_{i \in S^1_j} (c_{ij} - c_{i\bar{j}}).$$

After facility $\bar{j}$ was closed in move $k - 1$, $\Delta z^k_j$ for each $j \in J_0$ (19) can be rewritten as

$$\Delta z^k_j = f_j + \sum_{i \in (P_j - S^1_j)} (c_{ij} - c_{i\bar{d}}) + \sum_{i \in S^1_j} (c_{ij} - c_{i\bar{j}}) + \sum_{i \in S^2_j} (c_{ij} - c_{i\bar{j}}).$$

Therefore, we have

$$\Delta z^k_j - \Delta z^{k-1}_j = \sum_{i \in S^1_j} (c_{ij} - c_{i\bar{d}}) + \sum_{i \in S^2_j} (c_{ij} - c_{i\bar{j}}) = \sum_{i \in S^1_j} (c_{ij} - c_{i\bar{d}}) + \sum_{i \in S^2_j} (c_{ij} - c_{i\bar{j}}).$$

For each $i \in S^1_j \cup S^2_j$, $d^1_i$ is updated to $d^1_i = d^2_i$ and $d^2_i$ is determined through (9) after $\Delta z^k_j$ is updated.

3.2.4. Open facilities after another facility closed

Suppose facility $\bar{j}$ was closed in move $k - 1$. Define two sets $T^1_j$ and $T^2_j$ for each $j \in J_1$ as follows:

$$T^1_j = \{ i | d^1_i = j \land d^2_i = \bar{j} \} \quad \text{and} \quad T^2_j = \{ i | d^1_i = j \land d^2_i = \bar{j} \}. \quad (29)$$

In this case, define $d^3_i$ for each $i \in T^1_j \cup T^2_j$ as the index of the facility such that

$$c_{i\bar{d}}^3 = \min\{ c_{ij} | j \in J_1 \land j \neq d^1_i \land j \neq d^2_i \}. \quad (30)$$

**Theorem 6.** After facility $\bar{j}$ is closed in move $k - 1$, $\Delta z^k_j$ is updated from $\Delta z^{k-1}_j$ for each $j \in J_1$ according to

$$\Delta z^k_j = \Delta z^{k-1}_j + \sum_{i \in T^1_j} (c_{i\bar{d}}^3 - c_{ij}) + \sum_{i \in T^2_j} (c_{i\bar{d}}^3 - c_{i\bar{j}}). \quad (31)$$
Proof. Before facility \( j \) was closed in move \( k - 1 \), \( \Delta z_j^{k-1} \) for each \( j \in J_1 \) (21) can be rewritten as

\[
\Delta z_j^{k-1} = -f_j + \sum_{i \in (Q_j - T_j^2)} (c_{id_i^3} - c_{ij}) + \sum_{i \in T_j^2} (c_{ij} - c_{ij}).
\]

After facility \( j \) was closed in move \( k - 1 \), \( \Delta z_j^k \) for each \( j \in J_1 \) (21) can be rewritten as

\[
\Delta z_j^k = -f_j + \sum_{i \in (Q_j - T_j^2)} (c_{id_i^3} - c_{ij}) + \sum_{i \in T_j^1} (c_{id_i^3} - c_{ij}) + \sum_{i \in T_j^2} (c_{id_i^3} - c_{ij}).
\]

As a result, we have

\[
\Delta z_j^k - \Delta z_j^{k-1} = \sum_{i \in T_j^1} (c_{id_i^3} - c_{ij}) + \sum_{i \in T_j^2} (c_{id_i^3} - c_{ij}) - \sum_{i \in T_j^2} (c_{ij} - c_{ij})
\]
\[
= \sum_{i \in T_j^1} (c_{id_i^3} - c_{ij}) + \sum_{i \in T_j^2} (c_{id_i^3} - c_{ij}). \quad \square
\]

After \( \Delta z_j^k \) is updated, \( d_1^i \) and \( d_2^i \) are updated to \( d_1^i = d_2^i = d_3^i \) for each \( i \in T_j^1 \) and \( d_2^i = d_3^i \) for each \( i \in T_j^2 \). If facility \( j \) is the only one such that \( j \in J_1 \) after facility \( \bar{j} \) is closed, then the \( d_3^i \)'s are not defined, and \( \Delta z_j^k \) is not defined and cannot be updated using (31). In this case, we set \( \Delta z_j^k = \infty \) to ensure that facility \( j \) is not chosen to close in the next move.

4. An example

In this section, an example is presented to demonstrate how the \( \Delta z_j^k \)'s are computed initially and how they are updated thereafter. However, it does not show the steps in the TS procedure.

4.1. The example problem

A UFL problem with \( m = 15 \) and \( n = 8 \) is used in this example. The values of the \( c_{ij} \)'s are listed in the body and the values of the \( f_j \)'s are listed at the bottom of Table 1.

In the local optimal solution obtained at the end of the DROP greedy method, \( J_0 = \{1, 2, 5, 6, 8\} \) and \( J_1 = \{3, 4, 7\} \). The indices \( d_1^i \) and \( d_2^i \) for each customer \( i \) at this local optimal solution are listed in Table 2. The total cost of this solution is 692.

4.2. Computation of net cost changes

Using (18) and (20), the set \( P_j \) for each \( j \in J_0 \) and the set \( Q_j \) for each \( j \in J_1 \) are listed in Table 3. The values of \( \Delta z_j^k \) computed with (19) for each \( j \in J_0 \) and with (21) for each \( j \in J_1 \) are also listed in Table 3. Because the current solution is locally optimal, \( \Delta z_j^k > 0 \) for all \( j \in J \).
Table 1
Cost coefficients in the example problem

<table>
<thead>
<tr>
<th>Customer (i)</th>
<th>Facility (j)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>60</td>
<td>84</td>
<td>16</td>
<td>53</td>
<td>54</td>
<td>20</td>
<td>88</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>86</td>
<td>38</td>
<td>44</td>
<td>33</td>
<td>62</td>
<td>19</td>
<td>89</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>89</td>
<td>88</td>
<td>23</td>
<td>91</td>
<td>85</td>
<td>28</td>
<td>65</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>45</td>
<td>80</td>
<td>94</td>
<td>76</td>
<td>27</td>
<td>29</td>
<td>17</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>47</td>
<td>70</td>
<td>96</td>
<td>99</td>
<td>39</td>
<td>79</td>
<td>30</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>45</td>
<td>69</td>
<td>25</td>
<td>96</td>
<td>61</td>
<td>97</td>
<td>21</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>41</td>
<td>18</td>
<td>59</td>
<td>18</td>
<td>42</td>
<td>28</td>
<td>72</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>95</td>
<td>31</td>
<td>53</td>
<td>48</td>
<td>75</td>
<td>88</td>
<td>85</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>53</td>
<td>38</td>
<td>67</td>
<td>24</td>
<td>47</td>
<td>21</td>
<td>11</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>39</td>
<td>25</td>
<td>47</td>
<td>50</td>
<td>80</td>
<td>77</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>93</td>
<td>81</td>
<td>37</td>
<td>10</td>
<td>32</td>
<td>98</td>
<td>36</td>
<td>96</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>74</td>
<td>80</td>
<td>30</td>
<td>56</td>
<td>81</td>
<td>87</td>
<td>48</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>52</td>
<td>64</td>
<td>43</td>
<td>92</td>
<td>57</td>
<td>56</td>
<td>78</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>25</td>
<td>63</td>
<td>48</td>
<td>16</td>
<td>21</td>
<td>62</td>
<td>41</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>40</td>
<td>97</td>
<td>38</td>
<td>17</td>
<td>37</td>
<td>10</td>
<td>97</td>
<td></td>
</tr>
</tbody>
</table>

\( f_j \)

119 114 117 108 128 144 116 124

Table 2
The indices \( d_1^i \) and \( d_2^i \) for each customer \( i \) at the local optimal solution

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1^i )</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>( d_2^i )</td>
<td>4</td>
<td>3</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 3
The elements of \( P_j \) or \( Q_j \) and the values of \( \Delta z_j^1 \) for each facility at the local optimal solution

<table>
<thead>
<tr>
<th>( j )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_j ) or ( Q_j )</td>
<td>10</td>
<td>8</td>
<td>1, 3, 10, 12, 13</td>
<td>2, 7, 8, 11, 14</td>
<td>2</td>
<td>4, 5, 6, 9, 15</td>
<td>2, 6, 8, 13</td>
<td></td>
</tr>
<tr>
<td>( \Delta z_j^1 )</td>
<td>118</td>
<td>97</td>
<td>37</td>
<td>0</td>
<td>128</td>
<td>130</td>
<td>54</td>
<td>49</td>
</tr>
</tbody>
</table>

As examples, because \( 1 \in J_0 \), using (19) we obtain

\[
\Delta z_1^1 = f_1 + (c_{10,1} - c_{10,3}) = 119 + (24 - 25) = 118,
\]
After the values of the $d$’s are updated, indices $j$’s are updated with (27)–(31). The elements of the sets $S_j$ for each $j$ in $J_1$ are updated using (28) for each $j$ in $J_0$ and using (31) for each $j$ in $J_1$ are also listed in Table 4.

4.3. Update of net cost changes

Starting the short term memory process, facility $\bar{j} = 4$ is selected to switch status at the first move. Because facility 4 was originally open and is closed at the first move, the $\Delta z_j^2$’s are updated with (27)–(31). The elements of the sets $S_j^1$ and $S_j^2$ for each $j \in J_0$ (27) and of the sets $T_j^1$ and $T_j^2$ for each $j \in J_1$ (29) are listed in Table 4. The values of $\Delta z_j^2$ updated using (28) for each $j \in J_0$ and using (31) for each $j \in J_1$ are also listed in Table 4.

To compute $\Delta z_j^2$ for each $j \in J_1$, we also need to determine a facility index $d_i^3$ for each $i \in T_1 \cup T_2$ as defined in (30). These indices for the current solution are listed in Table 5.

Table 4
The elements of $S_j^1$, $S_j^2$, $T_j^1$ or $T_j^2$ and the values of $\Delta z_j^2$ for each facility after the first move

<table>
<thead>
<tr>
<th>$j$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_j^1$ or $T_j^1$</td>
<td>8</td>
<td>2, 7, 8</td>
<td></td>
<td></td>
<td>2</td>
<td>11, 14</td>
<td>2, 8</td>
<td></td>
</tr>
<tr>
<td>$S_j^2$ or $T_j^2$</td>
<td>7, 14</td>
<td>2, 7</td>
<td>1, 10</td>
<td></td>
<td>7, 11, 14</td>
<td>7</td>
<td>4, 9, 15</td>
<td></td>
</tr>
<tr>
<td>$\Delta z_j^2$</td>
<td>84</td>
<td>45</td>
<td>192</td>
<td>0</td>
<td>87</td>
<td>88</td>
<td>182</td>
<td>33</td>
</tr>
</tbody>
</table>

Table 5
The index $d_i^3$ for each customer after the first move

<table>
<thead>
<tr>
<th>$i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_i^3$</td>
<td>7</td>
<td>7</td>
<td>/</td>
<td>3</td>
<td>/</td>
<td>/</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>/</td>
<td>/</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

and because $3 \in J_1$, using (21) we obtain

$$
\Delta z_3^1 = -f_3 + (c_{14} - c_{13}) + (c_{37} - c_{33}) + (c_{10,4} - c_{10,3}) + (c_{12,7} - c_{12,3}) + (c_{13,7} - c_{13,3})
$$

$$
$$

As an example, because $2 \in J_0$, using (28) we have

$$
\Delta z_2^2 = \Delta z_2^1 + (c_{84} - c_{83}) + (c_{22} - c_{23}) + (c_{72} - c_{73})
$$

$$
$$

As a second example, because $3 \in J_1$, using (31) we have

$$
\Delta z_3^2 = \Delta z_3^1 + (c_{27} - c_{23}) + (c_{77} - c_{73}) + (c_{87} - c_{83}) + (c_{17} - c_{14}) + (c_{10,7} - c_{10,4})
$$

$$
$$

After the values of the $\Delta z_j^2$’s are updated, indices $d_i^1$ and $d_i^2$ for some $i$ are also updated. These indices with the updates are shown in Table 6.

Facility $\bar{j} = 8$ is selected to switch status in the second move. Because facility 8 was originally closed and is opened in the second move, the $\Delta z_j^3$’s are updated using (23)–(26). The elements of the sets $S_j^1$ and
Table 6
The indices \(d_1^i\) and \(d_2^i\) for each customer after the first move

<table>
<thead>
<tr>
<th>(i)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1^i)</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>(d_2^i)</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 7
The elements of \(S_1^j\), \(S_2^j\), \(T_1^j\) or \(T_2^j\) and the values of \(\Delta z_3^j\) for each facility after the second move

<table>
<thead>
<tr>
<th>(j)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>(S_1^j) or (T_1^j)</td>
<td>2, 8, 13</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(S_2^j) or (T_2^j)</td>
<td>2, 8</td>
<td>1, 3, 10, 12</td>
<td>2, 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\Delta z_3^j)</td>
<td>84</td>
<td>73</td>
<td>−9</td>
<td>16</td>
<td>87</td>
<td>103</td>
<td>30</td>
<td>−33</td>
</tr>
</tbody>
</table>

Table 8
The indices \(d_1^i\) and \(d_2^i\) for each customer after the second move

<table>
<thead>
<tr>
<th>(i)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1^i)</td>
<td>3</td>
<td>8</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>(d_2^i)</td>
<td>8</td>
<td>2</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>8</td>
<td>3</td>
</tr>
</tbody>
</table>

\(S_j^2\) for each \(j \in J_0\) (23) and those of \(T_j^1\) and \(T_j^2\) for each \(j \in J_1\) (25) are listed in Table 7. The values of the \(\Delta z_3^j\)'s, also listed in Table 7, are updated using (24) for each \(j \in J_0\) and using (26) for each \(j \in J_1\).

For instance, because \(2 \in J_0\), \(\Delta z_3^2\) is updated from \(\Delta z_3^2\) using (24) as

\[
\Delta z_3^2 = \Delta z_3^2 + (c_{23} - c_{22}) + (c_{83} - c_{82}) = 45 + (44 - 38) + (53 - 31) = 73.
\]

Because \(3 \in J_1\), \(\Delta z_3^3\) is updated from \(\Delta z_3^2\) using (26) as

\[
\Delta z_3^3 = \Delta z_3^2 + (c_{23} - c_{27}) + (c_{83} - c_{87}) + (c_{13.3} - c_{13.7}) + (c_{18} - c_{17})
+ (c_{38} - c_{37}) + (c_{10.8} - c_{10.7}) + (c_{12.8} - c_{12.7})
\]

After the values of the \(\Delta z_3^j\)'s are updated, the indices \(d_1^i\) and \(d_2^i\) for some \(i\) are also updated. These indices with the updates are given in Table 8.
5. Computational experiments

Computational experiments were conducted to test the performance of the TS procedure against the Lagrangian method, [3,21]. The TS and CLM methods developed by Ghosh [25], and the Hybrid method developed by Resende and Werneck [27]. The methods of Ghosh [25] are used because no comparative study with these methods has been reported. Hybrid is used because Resende and Werneck [27] reported that it outperformed the TS method of Michel and Van Hentenryck [26] that in turn was found to be the most effective method by Hoefer [41]. For notational convenience in this section, GTS is used to denote the TS procedure developed by Ghosh [25] and TS is used to denote the heuristic procedure developed in this study. TS was implemented in Fortran and the Fortran code of the Lagrangian method [3,21] implemented by Sun [28] is used in this study. These Fortran codes were compiled with the f77 compiler at optimization level 3 under the UNIX operating system. The C codes of GTS and CLM [25] and the C++ code of Hybrid [27] were obtained from the original authors. As Ghosh [25] did, we solved each test problem with CLM first and then with GTS allowing the same amount of CPU time as used by CLM with one exception for the Kochetov [42] problems. All computations were conducted on a SUN Enterprise 3000 computer.

Test problems both from the literature [25,35,42,43] and randomly generated are used in the computational experiments. Problems from the literature provide a benchmark to measure the performance of TS relative to other existing methods. Randomly generated test problems with different properties may represent a wider range of real life applications. Some of the randomly generated test problems were used by Sun [28].

If not specifically mentioned, the default parameter values used in TS are $l_1^1 = l_1^0 = 10$, $l_2^1 = l_2^0 = 20$, $C = 5$, $z_1 = 2.5$ and $z_2 = 0.5$. The best solution could be found with a smaller value of $C$ and smaller values of $z_1$ and $z_2$ for most of the test problems. In the following tables, $F$ is used to denote the number of times the long term memory process had been invoked before the best solution was found the first time by TS. A $F = 0$ means the best solution was found before the long term memory process was invoked the first time. We use $N$ to denote the number of search cycles in which the best solution is found with TS. The value of $N$ is reported if computational results on individual problems are presented and the average value of $N$ is reported if only summaries on a type of problems are presented.

5.1. Results of test problems from the literature

Four sets of test problems from the literature were used. One set was from the OR-Library [35], one set was provided by Ghosh [25], one set was created by Kratica et al. [43], and the other set was created by Kochetov [42].

5.1.1. Test problems from the OR-Library

There are currently 15 UFL test problems in the OR-Library [35]. Among these test problems, 12 are relatively small in size ranging from $m \times n = 50 \times 16$ to $m \times n = 50 \times 50$. The other three are relatively large with $m \times n = 1000 \times 100$. Computational results of these problems are reported in Table 9. Each row of the table gives the results of one individual problem. The problem names are originally used in the OR-Library.
Table 9
Computational results of test problems from the OR-Library

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Frequency</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F</td>
<td>N</td>
</tr>
<tr>
<td>Cap71</td>
<td>50 × 16</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap72</td>
<td>50 × 16</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap73</td>
<td>50 × 16</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap74</td>
<td>50 × 16</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap101</td>
<td>50 × 25</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Cap102</td>
<td>50 × 25</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap103</td>
<td>50 × 25</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap104</td>
<td>50 × 25</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Cap131</td>
<td>50 × 50</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Cap132</td>
<td>50 × 50</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Cap133</td>
<td>50 × 50</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Cap134</td>
<td>50 × 50</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Capa</td>
<td>1000 × 100</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Capb</td>
<td>1000 × 100</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Capc</td>
<td>1000 × 100</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

TS found optimal solutions for all these test problems without any exception. The optimal solution had been found for all, but one, of the 12 small problems before the long term memory process was invoked once. Because the optimal solution was found very early, TS could use substantially less CPU time without missing the optimal solution for any problem by using a small value of \( C \), such as \( C = 3 \).

Given \( C = 5 \), the maximum possible value of \( N \) is 6. The optimal solution was found more frequently for smaller problems. Optimal solutions were also easily obtained by TS using different ranges of \( l_c \) and \( l_o \) and different values of \( C, z_1 \) and \( z_2 \). The Lagrangian method found optimal solutions for all, but the last, of these problems. Results in the last two columns of Table 9 show that these two methods take approximately the same amount of CPU time for each problem.

The greedy method embedded in TS found optimal solutions for 6 out of the 12 small problems. As part of their computational experiment, Grolimund and Ganascia [24] used these small problems to test their TS procedures. Using several CPU hours for each problem, their most effective procedure obtained solutions 14%, while their least effective procedure obtained solutions 34%, worse than the optimal solutions on average. Their article did not report computational results on the three relatively large problems. Al-Sultan and AL-Fawzan [23] also found optimal solutions for the 12 small test problems with their TS procedure. They did not report results on the three larger problems either. Hybrid also found optimal solutions for all these problems [27].

5.1.2. Test problems from Ghosh [25]

The test problems with \( m \times n = 250 \times 250, m \times n = 500 \times 500 \) and \( m \times n = 750 \times 750 \) from Ghosh [25] are now available on the Internet [36]. Test problems of the same size are divided into two categories, symmetric, where \( c_{ij} = c_{ji} \) holds, and asymmetric, where \( c_{ij} = c_{ji} \) does not necessarily hold. For all problems \( c_{ij} \) is uniformly chosen from the interval [1000, 2000]. Problems in each category are further
Table 10
Computational results of test problems from Ghosh [25]

<table>
<thead>
<tr>
<th>Problem</th>
<th>Fixed cost range</th>
<th>Cost</th>
<th>N</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Type</td>
<td>TS</td>
<td>GTS</td>
<td>CLM</td>
</tr>
<tr>
<td>(m = n)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>Symmetric</td>
<td>257805.0</td>
<td>257832.6</td>
<td>257895.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>257917.8</td>
<td>257978.4</td>
<td>258032.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symmetric</td>
<td>276035.2</td>
<td>276185.2</td>
<td>276352.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>276053.2</td>
<td>276467.2</td>
<td>276184.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symmetric</td>
<td>333671.6</td>
<td>333820.0</td>
<td>333671.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>332897.2</td>
<td>333237.6</td>
<td>333058.4</td>
</tr>
<tr>
<td>500</td>
<td>Symmetric</td>
<td>511180.4</td>
<td>511383.6</td>
<td>511487.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>537912.0</td>
<td>538480.4</td>
<td>538685.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symmetric</td>
<td>621059.2</td>
<td>621107.2</td>
<td>621172.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>511140.0</td>
<td>511251.6</td>
<td>511393.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symmetric</td>
<td>537847.6</td>
<td>538144.0</td>
<td>538421.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>621463.8</td>
<td>621811.8</td>
<td>621990.8</td>
</tr>
<tr>
<td>750</td>
<td>Symmetric</td>
<td>763693.4</td>
<td>763830.8</td>
<td>763978.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>796571.8</td>
<td>796919.0</td>
<td>797173.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symmetric</td>
<td>900158.6</td>
<td>901158.4</td>
<td>900785.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>763717.0</td>
<td>763836.6</td>
<td>764019.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Symmetric</td>
<td>796374.4</td>
<td>796859.0</td>
<td>796754.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asymmetric</td>
<td>900193.2</td>
<td>900514.2</td>
<td>900349.8</td>
</tr>
</tbody>
</table>

The results of these test problems are reported in Table 10. Following Ghosh [25], we reported the average total cost of the five problems of each problem type obtained with TS, GTS, CLM and Hybrid. CLM obtained the same solutions for all problem types as reported by Ghosh [25]. However, GTS found better solutions for some problem types. The differences were caused by the time functions in C when measuring the CPU times. Solutions better than those reported by Ghosh [25] were highlighted in Table 10. Hybrid found solutions similar to those reported by Resende and Werneck [27]. For all problem types, TS found solutions better than both of those of GTS and CLM with only one exception where TS and CLM found solutions with the same average total cost for symmetric problems with \(m \times n = 250 \times 250\) in type C. TS also found better solutions than Hybrid did for 10 out of 18 problem types and found the same solutions for the rest while using about the same amount of CPU time. As shown in Table 10, the average CPU time used by GTS and CLM for each problem type is from 5 to 100 times of that used by TS and 4 to 75 times of that used by Hybrid.
These test problems, called the M* problems in the literature, are also available on the Internet [36]. The differences among the six sets of problems are in their sizes. The MO*, MP*, MQ*, MR*, MS* and MT* problems have $m \times n = 100 \times 100, 200 \times 200, 300 \times 300, 500 \times 500, 1000 \times 1000$ and $2000 \times 2000$, respectively. Optimal solutions for the first three sets are known. According to Kratica et al. [43], Hoefer [41] and Michel and Van Hentenryck [26], these problems are supposed to be very challenging. Their computational results are reported in Table 11.

With $C = 2$, TS found exactly the same solutions as Hybrid did. It found optimal solutions for all problems in the first three sets and found the best upper bounds for the next two sets. For problem MT1, the solution it found is better than the known bound. It found the best solution even without invoking the long term memory process for 19 out of the 22 problems as shown by the values of $F$. It also found the best solution frequently in the different search cycles as shown by the values of $N$. The Lagrangian method also found very good solutions. Evidently, these problems are not difficult to solve. For large problems, TS used more CPU time than Hybrid did. For problems with a large $n$, TS may need more CPU time because it updates the net cost change for every facility after a move.
5.1.4. Test problems from Kochetov [42]

For a given value of \( \kappa \), the values of \( m \) and \( n \) for these problems are determined by \( m = n = \kappa^2 + \kappa + 1 \). The fixed costs are \( f_j = 3000 \) for all \( j \). For any given \( j \) or \( i \), exactly \( \kappa + 1 \) values of the \( c_{ij} \)'s are chosen from the set \{0, 1, 2, 3, 4\} and the rest are \( \infty \). Two sets, each with 30 problems, with \( \kappa = 11 \) and \( \kappa = 17 \) or \( m \times n = 133 \times 133 \) and \( m \times n = 307 \times 307 \), called FPP11 and FPP17, respectively, are available. As showed by Resende and Werneck [27], these problems are very difficult to solve. These problems and their optimal solutions are available on the Internet [36,42].

For GTS and CLM, the CPU time for each problem is limited to 100 s for \( m \times n = 133 \times 133 \) problems and to 500 s for \( m \times n = 307 \times 307 \) problems. However, CLM stopped without reaching the CPU time limit for any problem. Results for these problems are reported in Tables 12 and 13. Because the optimal solution \( \hat{z} \) of each problem is known, \( \text{gap} \) is used to measure solution quality, where \( \text{gap} \) is defined as

\[
\text{gap} = \frac{z - \hat{z}}{\hat{z}} \times 100.
\]

In the tables, the mean values of \( \text{gap} \) of each set of problems are reported. Table 12 compares the results obtained by different methods. Using the default parameter values, TS did not find satisfactory solutions for these problems. The Lagrangian method found much worse solutions using about the same amount of CPU time as TS did. CLM found slightly worse solutions than TS did by using from 20 to 30 times of CPU time. GTS found slightly better solutions than both TS and CLM did by using substantially more
Table 14
Computational results of problems with different sizes

<table>
<thead>
<tr>
<th>$m \times n$</th>
<th>Cost ratio ($q$)</th>
<th>Better solution</th>
<th>$N$</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GTS</td>
<td>CLM</td>
<td>Lagrange</td>
<td>GTS</td>
</tr>
<tr>
<td>$m \times n = 500 \times 50$</td>
<td>100.00</td>
<td>100.00</td>
<td>120.51</td>
<td>0</td>
</tr>
<tr>
<td>$m \times n = 1000 \times 100$</td>
<td>100.09</td>
<td>100.29</td>
<td>117.53</td>
<td>1</td>
</tr>
<tr>
<td>$m \times n = 1500 \times 150$</td>
<td>100.64</td>
<td>101.06</td>
<td>119.81</td>
<td>5</td>
</tr>
<tr>
<td>$m \times n = 2000 \times 200$</td>
<td>100.12</td>
<td>100.26</td>
<td>117.92</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 15
Computational results of problems with different $m$

<table>
<thead>
<tr>
<th>$m \times n$</th>
<th>Cost ratio ($q$)</th>
<th>Better solution</th>
<th>$N$</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GTS</td>
<td>CLM</td>
<td>Lagrange</td>
<td>GTS</td>
</tr>
<tr>
<td>$m \times n = 500 \times 100$</td>
<td>100.71</td>
<td>100.43</td>
<td>122.69</td>
<td>2</td>
</tr>
<tr>
<td>$m \times n = 1000 \times 100$</td>
<td>100.09</td>
<td>100.29</td>
<td>117.53</td>
<td>1</td>
</tr>
<tr>
<td>$m \times n = 1500 \times 100$</td>
<td>100.33</td>
<td>100.51</td>
<td>119.65</td>
<td>5</td>
</tr>
<tr>
<td>$m \times n = 2000 \times 100$</td>
<td>100.05</td>
<td>100.54</td>
<td>116.99</td>
<td>2</td>
</tr>
</tbody>
</table>

CPU time. Hybrid obtained much better solutions than TS did although using much more CPU time. The Hybrid solutions are very close to those reported by Resende and Werneck [27].

An optimal solution of such a problem has far more closed than open facilities. Too many facilities are restricted to be open and, therefore, it is hard for TS to reach any optimal solution if $l_0 = l_c$ is used. Hence, $l_0 < l_c$ must be used for TS to be effective. By using $l_1^0 = 3$, $l_2^0 = 4$, $C = 50$, $z_1 = 1.0$ and $z_2 = 0.2$ and changing the values of $l_1^c$ and $l_2^c$, TS easily obtained near optimal solutions by using slightly less CPU time than Hybrid did. Sample solutions are presented in Table 13. In the column “Optimal”, the number of problems for which TS found an optimal solution is reported. Very good solutions were also obtained by using different TS parameter values as long as both the values of $l_1^0$ and $l_2^0$ are not too large. Searching more extensively by using several times of CPU time, Hybrid also obtained such good solutions [27].

5.2. Results of randomly generated test problems

Randomly generated test problems differ in four characteristics, problem size, problem shape, cost coefficient ranges, and cost coefficient scales. Problems with the same characteristics form a problem type. Computational results on 22 problem types, each with 15 problems, are reported in Tables 14–19. With one exception, TS and Hybrid found best solutions with exactly the same quality for each problem using approximately the same amount of CPU time. For one problem, the TS solution is better than the Hybrid solution. As a result, the Hybrid CPU times, but not solution quality, are reported in the following tables.
Table 16
Computational results of problems with different $n$

<table>
<thead>
<tr>
<th>$m \times n$</th>
<th>Cost ratio ($q$)</th>
<th>Better solution</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GTS</td>
<td>CLM</td>
<td>Lagrange</td>
</tr>
<tr>
<td>$m \times n = 1000 \times 50$</td>
<td>100.07</td>
<td>100.05</td>
<td>118.04</td>
</tr>
<tr>
<td>$m \times n = 1000 \times 100$</td>
<td>100.09</td>
<td>100.29</td>
<td>117.53</td>
</tr>
<tr>
<td>$m \times n = 1000 \times 150$</td>
<td>100.86</td>
<td>100.39</td>
<td>120.11</td>
</tr>
<tr>
<td>$m \times n = 1000 \times 200$</td>
<td>100.48</td>
<td>100.89</td>
<td>120.20</td>
</tr>
</tbody>
</table>

Table 17
Computational results of problems with different $c_{ij}$ ranges

<table>
<thead>
<tr>
<th>$c_{ij}$</th>
<th>Cost ratio ($q$)</th>
<th>Better solution</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GTS</td>
<td>CLM</td>
<td>Lagrange</td>
</tr>
<tr>
<td>$1200 \leq c_{ij} \leq 1800$</td>
<td>100.22</td>
<td>100.04</td>
<td>116.88</td>
</tr>
<tr>
<td>$1000 \leq c_{ij} \leq 2000$</td>
<td>100.09</td>
<td>100.29</td>
<td>117.53</td>
</tr>
<tr>
<td>$800 \leq c_{ij} \leq 2200$</td>
<td>100.37</td>
<td>100.07</td>
<td>123.51</td>
</tr>
<tr>
<td>$600 \leq c_{ij} \leq 2400$</td>
<td>101.51</td>
<td>101.78</td>
<td>125.05</td>
</tr>
<tr>
<td>$400 \leq c_{ij} \leq 2600$</td>
<td>100.39</td>
<td>100.31</td>
<td>123.80</td>
</tr>
</tbody>
</table>

Table 18
Computational results of problems with different $f_j$ ranges

<table>
<thead>
<tr>
<th>$f_j$</th>
<th>Cost ratio ($q$)</th>
<th>Better solution</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GTS</td>
<td>CLM</td>
<td>Lagrange</td>
</tr>
<tr>
<td>$14 \leq f_j \leq 16$</td>
<td>102.26</td>
<td>101.47</td>
<td>122.48</td>
</tr>
<tr>
<td>$12 \leq f_j \leq 18$</td>
<td>101.50</td>
<td>101.35</td>
<td>117.29</td>
</tr>
<tr>
<td>$10 \leq f_j \leq 20$</td>
<td>100.09</td>
<td>100.29</td>
<td>117.53</td>
</tr>
<tr>
<td>$8 \leq f_j \leq 22$</td>
<td>100.55</td>
<td>100.63</td>
<td>118.33</td>
</tr>
<tr>
<td>$6 \leq f_j \leq 24$</td>
<td>100.00</td>
<td>100.17</td>
<td>124.03</td>
</tr>
</tbody>
</table>

5.2.1. Test problem characteristics

Problem size is measured by $m \times n$. Four problem types are used with sizes varying from $500 \times 50$ to $2000 \times 200$ to test the effects of problem size. The value of $m$ is increased at an interval of 500 and that of $n$ at an interval of 50 from one problem type to the next. The cost coefficients of these test problems are in the ranges $1000 \leq c_{ij} \leq 2000$ and $10000 \leq f_j \leq 20000$. Results of these test problems are reported in Table 14.

Problem shape is measured by the relative values of $m$ and $n$. Two sets of problems, each with four problem types, are used to test the effects of problem shapes. In one set, the value of $n$ is fixed at $n = 100$ and the value of $m$ is increased from 500 to 2000 at an interval of 500. Results of this set are reported in Table 15. In the other set, the value of $m$ is fixed at $m = 1000$ and that of $n$ is increased from 50 to 200 at
Table 19
Computational results of problems with different $f_j$ magnitudes

<table>
<thead>
<tr>
<th>$f_j$</th>
<th>Cost ratio $(q)$</th>
<th>Better solution</th>
<th>N</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GTS</td>
<td>CLM</td>
<td>Lagrange</td>
<td>GTS</td>
</tr>
<tr>
<td>$6000 \leq f_j \leq 16000$</td>
<td>100.61 100.22 124.40</td>
<td>3 1</td>
<td>4.53</td>
<td>6.935 23.422 6.839</td>
</tr>
<tr>
<td>$8000 \leq f_j \leq 18000$</td>
<td>100.36 101.00 118.83</td>
<td>2 6</td>
<td>4.07</td>
<td>7.632 22.457 7.573</td>
</tr>
<tr>
<td>$10000 \leq f_j \leq 20000$</td>
<td>100.09 100.29 117.53</td>
<td>1 2</td>
<td>4.40</td>
<td>8.737 24.500 8.211</td>
</tr>
<tr>
<td>$12000 \leq f_j \leq 22000$</td>
<td>100.27 100.54 122.01</td>
<td>5 6</td>
<td>3.47</td>
<td>9.089 25.793 9.220</td>
</tr>
<tr>
<td>$14000 \leq f_j \leq 24000$</td>
<td>100.18 100.13 118.17</td>
<td>2 3</td>
<td>3.27</td>
<td>9.412 29.255 8.981</td>
</tr>
</tbody>
</table>

an interval of 50. Results of this set are reported in Table 16. The cost coefficients of these problems are in the ranges $1000 \leq c_{ij} \leq 2000$ and $10000 \leq f_j \leq 20000$.

Ranges of cost coefficients are measured by the differences between the upper and lower limits of $c_{ij}$ and $f_j$ within which $c_{ij}$ and $f_j$ were generated. Two sets of test problems, each with five problem types, were used to test the effects of cost coefficient ranges. In one set, the range of $f_j$ was fixed at $10000 \leq f_j \leq 20000$ while the range of $c_{ij}$ varied. Starting with $1200 \leq c_{ij} \leq 1800$, we subtracted 200 from the lower limit and added 200 to the upper limit of $c_{ij}$ from one problem type to the next. Results of these test problems are reported in Table 17. In the other set, the range of $c_{ij}$ was fixed at $1000 \leq c_{ij} \leq 2000$ while the range of $f_j$ varied. Starting with $14000 \leq f_j \leq 16000$, we subtracted 2000 from the lower limit and added 2000 to the upper limit of $f_j$ from one problem type to the next. Results of these problems are reported in Table 18. The size of these problems is fixed at $m \times n = 1000 \times 100$.

When the cost coefficient scales varied from one problem type to the next, the width of cost coefficient ranges stayed the same. Five types of problems were used to test the effects of cost coefficient scales. The values of $c_{ij}$ were fixed in the range $1000 \leq c_{ij} \leq 2000$ while the scales of $f_j$ varied. Starting with $6000 \leq f_j \leq 16000$, we added 2000 to both the lower and upper limits of $f_j$ from one problem type to the next. Results of these problems are presented in Table 19. The size of these problems is fixed at $m \times n = 1000 \times 100$.

5.2.2. Solution quality

For all these randomly generated problems, the solutions of TS completely dominate those of GTS and CLM [25] and those of the Lagrangian method. We use the cost ratio $q$ defined in (33) to measure the quality of the best solution obtained with TS relative to that obtained with another method for a given test problem,

$$q = \frac{z - z'}{z' - \bar{z}} \times 100.$$  

(33)

In (33), $\bar{z}$ is the lower bound obtained with the Lagrangian method on the total cost of the optimal solution, $z'$ is the total cost of the best solution found by TS, and $z$ is the total cost of the best solution found by a competitive method. If $z' = z$, then $q = 100$. For a given problem, the larger the value of $q$, the better the solution of TS than that of the competitive method. The average values of $q$ for each problem type are reported under the heading “Cost ratio $(q)$” in the tables separately for GTS, CLM and the Lagrangian method. The average values of $q$ for GTS and CLM are greater than 100 with only a few exceptions. Both
GTS and CLM have $q = 100$ for the problems with $m \times n = 500 \times 50$ in Table 14. GTS has $q = 100$ for the problems with $6000 \leq f_j \leq 24000$ in Table 18. Given the fact that the local optimal solution of a greedy method is not much worse than the global optimal, the differences in solution quality of these procedures are not trivial. The average values of $q$ at about 120 for the Lagrangian method for all problem types indicate that the TS, GTS, CLM and Hybrid solutions are all much better than those of the Lagrangian method.

Although GTS and CLM did not find a solution better than that of TS for a single test problem, they found equally good solutions for a large part of these problems. The number of problems for which the TS solutions are better than the GTS or the CLM solutions is presented under the heading “Better solution” in the tables. From these numbers we can see that TS finds better solutions more often for problems that are larger and more difficult to solve.

TS found the best solution multiple times for each problem within a single run as evidenced by the average value of $N$ for each problem type. We also tried different ranges for $l_c$ and $l_o$ and different values for $C_1, \alpha_1$ and $\alpha_2$. TS found almost the same solutions for these test problems with different combinations of these parameters.

5.2.3. CPU time

The average CPU time for each problem type is reported for TS, CLM and Hybrid in the tables. The CPU times used by different methods varied in similar patterns as the problem characteristics changed although different methods used different amounts of CPU times for a given problem type. The CPU times used by GTS and CLM are from about 2 to about 30 times of those used by TS or by Hybrid for different problem types.

As expected, larger problems took more CPU time than smaller problems as shown in Table 14. More CPU times are needed for problems with a fixed $n$ when $m$ increased as shown in Table 15 and for problems with a fixed $m$ when $n$ increased as shown in Table 16. Because the number of moves needed is more closely related to $n$ than to $m$ in the stopping criteria of the short term (13) and medium term (16) memory processes, TS should take more CPU time for problems with a relatively large $n$. For problems with a fixed range of $f_j$, the CPU time taken by TS decreases as the range of $c_{ij}$ increases as shown in Table 17. When the range of $c_{ij}$ increases, fewer and fewer customers need to switch facilities after a facility opens or closes and, therefore, less computation is needed when the net cost changes are updated. For problems with a fixed range of $c_{ij}$, the CPU time taken by TS decreases as the range of $f_j$ increases as shown in Table 18. As the range of $f_j$ increases, facilities change status less frequently and therefore less computation is needed. With one exception, the CPU times used by all methods increase when the scales of $f_j$ increase as shown in Table 19.

TS found the best solution for each problem very early in the solution process. In fact, the best solution was found before the long term memory process was invoked for about 90% (296 out of 330) of these randomly generated test problems. Therefore, it can still find the same best solutions but use much less CPU time if a smaller value of $C$ is used.

6. Conclusions

A TS procedure for the UFL problem is developed, implemented and tested. Its performance is tested against the Lagrangian method and heuristic procedures recently reported by Ghosh [25] and Resende...
and Werneck [27] using test problems from the literature and test problems randomly generated. TS found optimal solutions for all problems with known optimal solutions except for some of those from Kochetov [42]. For all test problems, the TS solutions either match or dominate those of the competitive methods.

In addition to applying the UFL model and the TS procedure to real life problems, developing more effective and efficient heuristic solution procedures appears to be a research direction. Future research in this direction may focus on the selection of TS parameter values to improve its performance. Using different diversification or intensification strategies may also deserve investigation. With modification, the TS procedure developed in this study may be extended to other facility location problems, such as the capacitated facility location problem and the single source capacitated facility location problem.

Acknowledgements

The author is very grateful to Dr. Diptesh Ghosh of the Indian Institute of Management, Ahmedabad, for providing his TS and CLM C codes for the computational experiments and to Dr. Mauricio G. C. Resende of the AT&T Labs Research for making the Hybrid C++ code available for this study. Preliminary results of this study were presented at the International Conference on Adaptive Memory and Evolution: Tabu Search and Scatter Search, Oxford, MS, March 7–11, 2001.

References