Optimizing a Ring-Based Private Line Telecommunication Network Using Tabu Search

Jiefeng Xu • Steve Y. Chiu • Fred Glover
Delta Technology, Inc., 1001 International Boulevard, Atlanta, Georgia 30354-1801
jiefeng.xu@delta-air.com
GTE Laboratories, Inc., 40 Sylvan Road, Waltham, Massachusetts 02254
schiu@gte.com
Graduate School of Business, University of Colorado at Boulder, Boulder, Colorado 80309-0419
fred.glover@colorado.edu

One of the private line network design problems in the telecommunications industry is to interconnect a set of customer locations through a ring of end offices so as to minimize the total tariff cost and provide reliability. We develop a Tabu Search method for the problem that incorporates long term memory, probabilistic move selections, hierarchical move evaluation, candidate list strategies and an elite solution recovery strategy. Computational results for test data show that the Tabu Search heuristic finds optimal solutions for all test problems that can be solved exactly by a branch-and-cut algorithm, while running about three orders of magnitude faster than the exact algorithm. In addition, for larger size problems that cannot be solved exactly, the tabu search algorithm outperforms the best local search heuristic currently available. The performance gap favoring Tabu Search increases significantly for more difficult problem instances.
(*Digital Data Service*; *Telecommunications Network Design*; *Traveling Salesman Problem*; *Tabu Search*; *Heuristic*)

1. Introduction

Digital Data Service (DDS) is a high-quality digital transport service in the telecommunications industry using permanent network connections and dedicated transmission facilities. In this paper, we address a particular DDS network design problem that is encountered by a major telecommunications company in the United States. The input elements of the problem include a set of end offices, a set of digital hubs, and a set of customer locations that are geographically distributed on a plane. Each customer location is connected directly to its own designated end office, which in turn needs to be connected to exactly one selected hub. Then the selected hubs must be connected by a ring. (The ring topology is widely used in communications network designs to provide reliability.) Each hub has a fixed cost for being chosen and each link has a connection cost for being included in the solution. The objective is to design such a network at minimum cost.

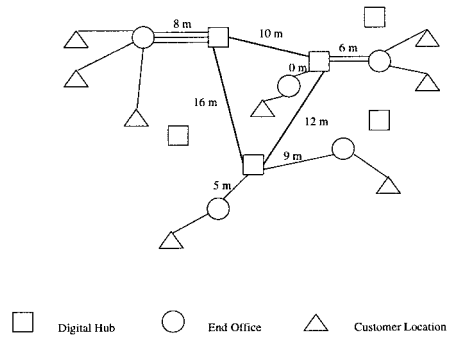
Figure 1 shows a real scenario of a small ring-based DDS network. The number of dedicated lines required for the link between an end office and its assigned hub is equal to the number of customer locations connected to the end office. The links between customer locations and end offices are not really part of the network design problem because they are uniquely

0025-1909/99/4503/0330$05.00

determined by the (nonoverlapping) serving areas of the end offices. Each customer location is always connected to its designated end office serving the area.

In practice, the link cost is sensitive to distance and is calculated according to the current tariff charges. These charges include a fixed cost and a variable cost per mile that both vary with the distance. For each active (selected) hub, the bridging cost is proportional to the number of lines connected to the hub. To illustrate how these costs are calculated, suppose the monthly cost data are given as follows:

Figure 1 A Ring-Based DDS Network



Fixed bridging cost:	$82.00		
Bridging cost per line:	$41.00		
Link cost:	Mileage	Fixed Cost	Variable Cost
	<1 mile	$30.00	$0.00
	1–15 miles	$125.00	$1.20
	≥16 miles	$130.00	$1.50.

Then the monthly costs for the network in Figure 1 are

Bridging Cost	
fixed cost:	$\$82.00 \times 3 = \$246.00$
variable cost:	$\$41.00 \times 14 = \$574.00$
Link Cost	
fixed cost:	$\$30.00 \times 1 + \$125.00 \times 9 + \$130.00 \times 1 = \$1285.00$
variable cost:	$\$1.20 \times (3 \times 8 + 10 + 2 \times 6 + 12 + 9 + 5) + \$1.50 \times 16 = \$106.40$
Total monthly cost:	$2211.40.

Note that a line connecting two active hubs has two bridging facilities at its ends, so it should be counted as twice in calculating the variable bridging cost. Consequently the decisions faced by the network designers are:

- Select a subset of hubs among all potential hubs and connect them via a ring (a travelling salesman tour over the selected hubs).
- Connect each end office to a selected hub (so that the original customer locations can communicate to each other).

The objective of the design is to minimize the total monthly cost as calculated in the above example.

In practice, the Federal Communications Commission (FCC) demands that telecommunication companies provide the best DDS design to customers. For real world instances, the number of customer locations (or end offices) can vary from 2 to over 100, and the number of potential hubs can be as large as 300. The algorithm reported in this paper is used in an automatic quoting system that requires the response time (solution time) to be within one minute, so that the sales representative can give the customer a quote over the phone. The challenge is to develop an algorithm that not only achieves such a response time, but that also provides optimal or near-optimal solutions for DDS design.

Throughout the paper, the hubs are referred to as *steiner nodes*, and the end offices are referred to as *target nodes*. Also notice that the cost for connecting a target node to a steiner node and the cost for connecting two steiner nodes can both be precalculated.

In this paper, we explore an implementation of Tabu Search (TS) for solving this ring-based DDS network design problem. TS is a metaheuristic that proves effective for many combinatorial optimization problems. For a comprehensive overview of TS, see Glover and Laguna (1997). In recent years, a growing number of TS applications have appeared in the area of telecommunications. Such applications include bandwidth packing (Laguna and Glover 1993), path assignment for dynamic routing (Anderson et al. 1993), SONET ring design (Laguna 1994), hub facility location (Skorin-Kapov and Skorin-Kapov 1994), digital line network design (Xu et al. 1996a, 1996b) and dynamic routing communication network design (Xu et al. 1997). The highly successful outcomes of these applications motivate us to develop and test a TS method designed specifically for the ring-based DDS network design problem.

This paper is organized as follows. We present the mathematical formulation in the next section. In §3 we describe a TS-based heuristic for the problem and examine several relevant issues such as long term memory, probabilistic move selection, neighborhood structure, hierarchical move evaluation and candidate list strategies. Section 4 reports computational results with two sets of carefully designed test problems, including comparisons with other exact and heuristic approaches. In the concluding section, we summarize our methodology and findings.

2. Mathematical Formulation

The problem addressed in this paper can be formulated as a 0-1 integer programming problem as follows. First the input data are:

M: set of target nodes;

N: set of steiner nodes;

c_{ij}: cost of connecting target node i to steiner node j;

d_{jk}: cost of connecting two steiner nodes j and k;

b_j: cost of using steiner node j.

The decision variables are:

x_{ij}: a binary variable equal to 1 if and only if target node i is linked to steiner node j;

y_{jk}: a binary variable equal to 1 if and only if steiner node j is linked to steiner node k $(j < k)$;

z_j: a binary variable equal to 1 if and only if steiner node j is selected to be *active*.

Then the formulation is

$$\text{minimize} \sum_{i\in M}\sum_{j\in N} c_{ij}x_{ij} + \sum_{j\in N}\sum_{\substack{k>j\\ k\in N}} d_{jk}y_{jk} + \sum_{j\in N} b_j z_j \quad (1)$$

$$\text{subject to:} \sum_{j\in N} x_{ij} = 1, \quad i \in M, \quad (2)$$

$$x_{ij} \le z_j, \quad i \in M,\ j \in N, \quad (3)$$

$$y_{jk} \le (z_j + z_k)/2, \quad j < k,\ j, k \in N, \quad (4)$$

$$\sum_{k\in N} y_{jk} + \sum_{k\in N} y_{kj} = 2z_j, \quad j \in N, \quad (5)$$

$$\sum_{j\in H}\sum_{k\in H} y_{jk} \le \sum_{j\in\{H-l\}} z_j + 1 - z_t,\ l \in H,\ H \subset N,\ |H| \ge 3,\ t \in N - H, \quad (6)$$

$$x_{ij} \in \{0, 1\}, \quad i \in M,\ j \in N, \quad (7)$$

$$y_{jk} \in \{0, 1\}, \quad k > j,\ j, k \in N, \quad (8)$$

$$z_j \in \{0, 1\}, \quad j \in N. \quad (9)$$

In this formulation, the objective function (1) seeks to minimize the sum of the connection cost between target nodes and steiner nodes, the connection cost between steiner nodes, and the setup cost for the steiner nodes. Constraint (2) specifies that each target node must be connected to exactly one steiner node. Constraint (3) indicates that the target nodes can only be connected to the active steiner nodes. Constraint (4) stipulates that two steiner nodes can be connected if and only if both nodes are active. Constraints (5) and (6) express the ring (or tour) structure over the active steiner nodes. In particular, (5) specifies the condition that each active steiner node must have a degree of two, while (6) is an *subtour-eliminating* constraint that compels all active steiner nodes to form a single tour. Finally, all decision variables are defined as binary.

Clearly, the ring-based DDS problem is NP-hard since the well-known Traveling Salesman Problem

(TSP) can be easily reduced to it. (For a complete review of the TSP, we refer readers to the two books (Lawler et al. 1985 and Reinelt 1994) and the two algorithm surveys by Laporte (1992) and Johnson and McGeoch (1996).) Only small size ring-based DDS problems (e.g., with 10 target nodes and 30 steiner nodes) can be solved exactly within a reasonable time period for the requirements of this application (i.e., one minute) by *state-of-the-art* integer programming techniques such as branch-and-cut. In fact, this holds true for a specialized branch-and-cut method based on the foregoing mathematical formulation which is tailored to generate constraints (or cuts) of type (6) on a needed basis. (See Lee et al. 1996a and 1996b.)

3. The Tabu Search Heuristic

Tabu Search is an aggressive search procedure that proceeds iteratively from one solution to another by moves in a neighborhood space with the assistance of *adaptive* memory. To exploit this memory effectively, the method makes use of several key strategic principles and associated algorithm designs. In this section, we first introduce an elementary TS heuristic, then describe each of the customary and more advanced components developed for the ring-based DDS problem.

3.1. Elementary Tabu Search Procedure

Tabu Search is an iterative method which can be used to guide traditional local search methods to escape the trap of local optimality. TS operates through neighborhood moves, that proceed from one solution to another at each iteration. Some moves are marked *Tabu* and are forbidden unless they lead to highly desirable outcomes. Let x_now be the solution at the current iteration, and x_best the best solution found so far, *iter* the current iteration counter, and $Tabu(iter)$ the set of Tabu moves at iteration *iter*. We define a move to be *admissible by aspiration* if it belongs to Tabu(iter), but if the solution produced by the move has a sufficiently high quality to allow its tabu status to be disregarded. A simplified (short-term memory) version of TS may be expressed as follows.

Step 0. $iter = 0$; Initialize x_now; $x\_best = x\_now$; $Tabu(iter) = \varnothing$.

Step 1. Construct a list of candidate moves from the neighborhood of x_now. Evaluate each candidate move.

Step 2. Select the highest evaluation move that does not belong to $Tabu(iter)$, or which qualifies to be selected as a result of being admissible by aspiration. Perform the move, and update x_now.

Step 3. If x_now is better than x_best, update x_best.

Step 4. If stopping criteria are satisfied, terminate with x_best. Otherwise, $iter = iter + 1$; update $Tabu(iter)$; go to *Step* 1.

Numerous advanced strategies exist that can effectively enhance this rudimentary short-term memory form of tabu search (see Glover and Laguna 1997). To illustrate our TS approach for this network design problem, the following subsections describe the issues of neighborhood structure and moves, memory structures, hierarchical move evaluations and candidate lists, probabilistic move selection and advanced intensification strategies.

3.2. Neighborhood Structure and Moves

We partition the steiner nodes into the disjoint subsets of active nodes (A) and inactive nodes ($\bar{\mathsf{A}}$). The moves that define the neighborhood structure for our procedure consist of transferring a chosen node from one of these two subsets to another, and of exchanging two nodes between these subsets. Specifically, we divide the transfer moves into the following two elementary types:

(1) Constructive move: transfer a selected steiner node from $\bar{\mathsf{A}}$ to A. This move inserts a node into the current TSP tour, and therefore increases the cardinality of the set A by one. This move is disallowed if the set $\bar{\mathsf{A}}$ is empty.

(2) Destructive move: transfer a steiner node from A to $\bar{\mathsf{A}}$. This move deletes the active steiner node from the current TSP tour, and therefore decreases the cardinality of the set A by one. This move is disallowed if the set A is empty.

Any set A can be reached via a sequence of constructive and/or destructive moves starting from any solution configuration. Thus, constructive and destructive moves are considered to be elementary moves in the search process. Pairwise exchange (swap) moves, which exchange one active steiner node with one inactive steiner node, can be viewed as a combination

of a constructive and a destructive move. Such a move leaves the cardinalities of both set A and $\bar{A}$ unchanged, but introduces a more significant change to the current TSP tour. The swap move is disallowed if either the set A or $\bar{A}$ is empty.

We observe that our simple set of fundamental moves is somewhat different from those customarily used in TSP applications. That is, while standard TSP heuristics may incorporate constructive steps (and Tabu Search variants also incorporate destructive steps), the exchanges used in such TSP heuristics are not the same as the exchanges we describe here. Our divergence from the classical choice of neighborhoods is motivated by the findings of Xu et al. (1996a, 1996b), which identified the current neighborhood structure to be highly effective when properly exploited, in a Tabu Search approach for a related class of telecommunication problems. In addition, we also make use of classical TSP neighborhoods, as noted later.

For a swap move evaluation, effort must be taken to reduce the computational expense when the number of steiner nodes is moderately large. For that purpose, a natural candidate list is constructed to isolate a promising subset of the swap moves. This candidate list restricts attention to pairs (x, y) whose elements are drawn from the K best destructive and constructive moves where K is an integer in the range of 5 to 15. This candidate list strategy is motivated in part by the idea of the Proximate Optimality Principle (POP) that says good solutions at one level are likely to be found close to good solutions at an adjacent level. (For example, we may conceive constructive and destructive moves as mechanisms for moving between levels, and swap moves as mechanisms for searching within a given level.) As a consequence, this candidate list is used to screen for the good partial moves whose composition may give a good candidate to evaluate. Such a candidate list strategy proves to be much faster than evaluating the whole swap neighborhood, yet can be implemented without sacrificing overall solution quality (see Xu et al. 1996a, 1996b).

We blend the elementary moves with the swap moves to produce the complete neighborhood search. Because a swap move involves a more significant change in the TSP tour (and hence requires a more complex evaluation of its consequences), we perform it more sparingly in the search process. In particular, we apply it chiefly in the roles of periodic perturbation and conditional oscillation. A perturbation step is guided by elementary moves and executed once for every certain number of iterations. The conditional oscillation step is designed to achieve a greater intensification of the search, by executing swap moves for some number of iterations when the search cannot improve the solution for a predefined duration. This mixed mechanism proves effective and efficient in our applications, since we find that a dominant reliance on the elementary moves, when handled intelligently, yields good decisions with only occasional reliance on more complex moves.

3.3. Tabu Search Memory

TS memory structures play a fundamental role in our algorithm to guide the search process. We use the short-term memory to prevent the search from being trapped in a local optimum and use the long-term memory to provide the diversification strategy.

Short-Term Tabu Search Memory. The short-term memory operates by imposing restrictions on the composition of new solutions generated (typically expressed as a restriction on *attributes* of these solutions). For elementary moves, we impose restrictions that assure a move cannot be "reversed." In particular, if the node x is currently dropped from the active steiner node set A, we forbid this node to move back to A for several iterations. For swap moves, we impose the restrictions on moves in both direction. If an active node x is swapped with an inactive node y in the current move, the restriction inhibits both moving node x back to A and moving node y back to $\bar{A}$. Such a restrictive mechanism prevents the search from revisiting a local optimum in the short term and greatly diminishes the chance of cycling in the long term.

How long a given restriction is in effect depends on a parameter called the *Tabu Tenure*, which identifies the number of iterations a particular Tabu restriction remains in force. The Tabu Tenure can be either fixed or variable, but a tenure that varies within a small range about a central value often proves more robust. Moreover, in our application, we allow the central

value to differ according to the move type. Since adding a node introduces a fixed cost, and thus makes the move appear less attractive than a destructive one, we assign a longer Tabu Tenure to avoid destructive moves than to avoid constructive moves.

A TS restriction may be overridden by means of *aspiration criteria* if the outcome of the move under consideration is sufficiently desirable. We use the simple criterion of overriding the restriction if the current candidate move would lead to a new best solution.

We implement the short-term memory using a recency-based memory structure as follows. Let iter denote the current iteration number, and let tabu_add(x) and tabu_drop(y) denote the future iteration values governing the duration that will forbid a reversal of the moves of adding node x and dropping node y (i.e. by preventing node x from being dropped and node y from being added). Similarly, let tabu_add_tenure and tabu_drop_tenure be the values of Tabu Tenures for these two moves. Initially, tabu_add(x) and tabu_drop(x) are set to zero for all nodes x, and iter starts at one. When the TS restriction is imposed, we update the recency memory as:

$$\text{tabu\_add}(x) = \text{iter} + \text{tabu\_add\_tenure}$$

(*for the constructive move of adding node x*),

$$\text{tabu\_drop}(y) = \text{iter} + \text{tabu\_drop\_tenure}$$

(*for the destructive move of dropping node y*).

Thus the restriction to prevent x from being dropped is enforced when tabu_add(x) $>$ iter, and the restriction to prevent y from being added is enforced when tabu_drop(y) $>$ iter. As previously noted, we select the central value for tabu_add_tenure to be smaller than that of tabu_drop_tenure. Let best_sol_cost be the cost of the best solution found so far, and best_move_cost be the evaluation (estimated cost) of the move we select. Also define cost(·) as the move evaluation value. Then the move selection procedure incorporating the TS restrictions and aspiration criteria proceeds as follows:

Assign a large value to best_move_cost.
For each inactive steiner node x, **do**
  if cost(x) $<$ best_move_cost **do**
    if cost(x) $<$ best_sol_cost **or** tabu_add(x) $\leq$ iter **do**
      best_move_cost = cost(x).
For each active steiner node y, **do**
  if cost(y) $<$ best_move_cost **do**
    if cost(y) $<$ best_sol_cost **or** tabu_drop(y) $\leq$ iter **do**
      best_move = cost(y).

For the exchange move, we have

Assign a large value to best_move_cost.
For each candidate node pair composed of inactive steiner node x **and active steiner node** y, **do**
  if cost(x, y) $<$ best_move_cost **do**
    if cost(x, y) $<$ best_sol_cost **or**
    (tabu_add(x) $\leq$ iter **and** tabu_drop(y) $\leq$ iter) **do**
      best_move_cost = cost(x, y).

Long Term Tabu Search Memory. The long-term TS memory we employ makes use of a frequency-based memory structure to achieve a diversification effect, encouraging the search to explore regions less frequently visited.

More specifically, we use this memory to discourage moves that occurred frequently during the search (and consequently to encourage moves that occurred less frequently). A transition measure is used to record the number of times each steiner node changes from an active status to an inactive status or vice versa. Let frequency0(x) be the number of times that steiner node x is changed from *active* to *inactive*, frequency1(x) be the number of times that steiner node x is changed from *inactive* to *active*. These frequencies can easily be updated as follows:

$$\text{frequency0}(x) = \text{frequency0}(x) + 1 \text{ if the move is destructive;}$$

$$\text{frequency1}(x) = \text{frequency1}(x) + 1 \text{ if the move is constructive.}$$

This transition measure is then normalized to lie in the interval [0, 1] by dividing by the maximum of frequency0(·) or frequency1(·) as appropriate. This normalized value is then linearly scaled by a selected constant to create a penalty term. The penalty term is added to the corresponding move evaluation so that the frequency factor is taken into account in the move selection procedure. It should be noted that this

long-term memory is designed strictly for diversification, without any counterbalancing consideration of intensification effects. A more advanced strategy would seek to integrate diversification and intensification issues, and we will examine such an integration in future research work.

3.4. Hierarchical Move Evaluation

Once the subset A is determined, the cost of the current solution can be calculated by: (1) constructing a minimum cost TSP tour over A and identifying the resulting cost; (2) linking every target node to its cheapest (i.e. cheapest-link) active steiner node and finding the sum of the resulting connection costs; and (3) summing all node costs (set-up costs) for A. The second part can be easily implemented by maintaining a presorted list for every target node, which records the connection costs from this target node to every steiner node. Thus, (2) can be found in linear time for each target node. The calculation of (3) is trivial. Therefore the key issue in the move evaluation becomes the TSP tour construction.

Since finding the optimal TSP tour is a NP-hard problem, it is not practical to use exact methods to evaluate the tour even when the number of nodes in the tour is moderate. Among the heuristics, some local search approaches such as 2-opt, 3-opt, or-opt, etc., work fast, but unless they are embedded in a design for going beyond local optimality, the solutions they obtain are often myopic. Metaheuristic approaches, which may incorporate simple heuristics within them, can overcome the limitation of the local search and can yield much better solutions, though typically at the expense of considerably more computation time. In our TS algorithm, we devise a hierarchical evaluation mechanism with the goal of achieving an effective tradeoff between the solution quality and the speed. This hierarchical evaluation employs the evaluators at three different levels (basic, intermediate, advanced), each associated with different types of neighborhood moves and appropriate candidate lists. The evaluators are based on identifing the cost of the corresponding TSP tour. (Note that this is not the full cost to be considered, since the costs of (2) and (3) must also be included in the complete evaluation of each move.) The higher level evaluator is more powerful and time-consuming than the lower level evaluator, and hence is applied more restrictively. We describe these evaluators as follows.

Basic Evaluator. The basic evaluator is used to evaluate every constructive, destructive and swap move in the candidate list. For constructive moves, the evaluator identifies the minimum insertion cost by inserting the new node into its *cheapest* insertion position. For destructive moves, the evaluator identifies the cost of removing the given node and simply connecting its two adjacent nodes in the current tour. For swap moves, the evaluator identifies the cost of first removing the given node and then inserting the new node as described above.

Intermediate Evaluator. The intermediate evaluator employs the 2-opt heuristic to improve the current tour. The 2-opt proceeds by considering all possible ways of removing two arcs from the current tour and then reconnecting the two resulting chains to form a new complete TSP tour. If the a new tour is found to be shorter than the current tour, then accept this tour and continue to proceed from this tour. The 2-opt terminates when no improvement can be obtained. The theoretical complexity of the 2-opt for finding the first improving move is $O(|A|^2)$, though in practice many implementation tricks can reduce this complexity significantly (see Johnson and McGeoch 1996).

The 2-opt procedure can be significantly simplified with our destructive and constructive moves. Suppose that the current tour is already a local optimum (e.g., improved by 2-opt), then the destructive move and the constructive move only introduce one and two new edges in the tour respectively. Therefore, the 2-opt needs to evaluate only the options that remove at least one of these new edges. The complexity of this simplified 2-opt for finding the first improving move is thereby reduced to $O(|A|)$. The changes brought by the swap moves can be exploited in a more complicated, but for simplicity we apply the standard 2-opt procedure for those tours since we do not execute the swap moves as frequently.

The intermediate evaluator is applied to a subset of selected neighborhood moves, that is, the candidate list maintained for the probabilistic move seletion (as

described in the next subsection), which consists of the non-Tabu neighborhood moves at the current iteration that have the K highest evaluations, based on the complete evaluation using the basic evaluator.

Advanced Evaluator. The advanced evaluator uses more complicated search teniques for improvement. First, it applies 3-opt local search to the current tour. The 3-opt application improves the tour by evaluting all possible ways of removing 3 arcs and reconnecting them to produce a new tour.

After the tour is improved by 3-opt, we employ a stand-alone simple TS algorithm for the TSP (TS-TSP). The TS-TSP uses simple ejection and swap moves applied to the nodes in the tour. A rudimentary short-term memory structure is used to discourage the search from revisiting previous solutions. At each iteration, the admissable move with the highest evaluation is selected and performed. The search terminates at a predetermined maximum number of iterations while the best solution over the entire search is recorded. The TS-TSP was first successfully used as a tour-improvement tool in the Vehicle Routing Problem (VRP) by Xu and Kelly (1996). Computational experience disclosed that the TS-TSP provides a simple approach to yield shorter TSP tours than 3-opt and significantly improves the search quality for the VRP.

Since the advanced evaluator is more complicated and time-consuming, we execute this evaluator on a more restrictive basis. The scenarioes where we run the advanced evaluator are: (1) when a "new best" solution is found; (2) when a current solution accumulates a certain degree of estimation error from the use of the intermediate evaluator; and (3) when the estimation errors of a set of "elite solutions" need to be corrected periodically.

The periodic correction in (3) seeks to balance the tradeoff between expected accuracy and speed of executing the algorithm. To achieve this, we manipulate a priority queue that includes a selected number of elite solutions encountered so far during the search, where these solutions consist of those actually visited and also of those that may potentially be visited by means of currently available candidate moves.

Before applying the error correction operation, the priority queue is ordered by the estimated costs (produced by intermediate evaluator) of its component solutions. Error correction using the advanced evaluator is then periodically performed on each element in this queue. Once an element's corrected cost is thus identified, this element is marked so that no error correction is executed on this element in the future. At the same time, the element is repositioned in the queue according to its new cost. Thus, when a new elite element is encountered whose estimated cost is better (smaller) than the cost of the current worst element of the queue, the new element is added and marked for error correction while the worst element is dropped from the queue. Because of periodic updating, the costs associated with queue elements can be a mix of estimated and corrected costs. The updating of the priority queue is further enhanced by applying a sorted pointer list to facilitate the add and drop operations.

Based on our empirical experience, the 2-opt based intermediate evaluator works quite well for tours containing 10 or fewer nodes. Thus, we do not bother performing the time-consuming advanced evaluator on those tours. Furthermore, since our TS algorithm generally starts from a poor solution (e.g., many solutions contain unnecessarily large number of active nodes), and this solution can be rapidly improved by our TS algorithm, there is no need to find the more accurate costs for these inferior solutions using the advanced evaluator. Consequently, we disable the advanced evaluator in the very early stage of the search.

Estimation errors can have a significant influence on move selection, especially for the large problem instances. To further compensate for the effects of approximation, we also use a move selection rule based on probabilistic Tabu Search, as described in the next subsection.

3.5. Probabilistic Move Selection

The fundamental idea of the move selection approach of probabilistic Tabu Search (Glover 1989) is simply to translate Tabu restrictions and aspirations into penalties and inducements that modify the standard evaluations, and then to map these modified evaluations into probabilities that are strongly biased to favor the highest evaluations. We are particularly motivated to

apply this approach in the present setting as a result of observations of Glover and Løkketangen (1996) concerning the uses of probabilities to combat "noise." Since we refine the candidate list and create the move evaluation based on a cost approximation, the move evaluation is contaminated by a form of noise, so that a "best evaluation" does not necessarily correspond to a "best move." Therefore we seek a way to assign probabilities that somehow compensates for the noise level.

We apply probabilistic Tabu Search in the following simple form.

Step 1 Generate the candidate list and evaluate the moves of this list, assigning penalties to moves that are tabu.

Step 2 Take the move from the candidate list with the highest evaluation value.

If the move satisfies the aspiration criterion, accept it and exit; otherwise, continue to *Step* 3.

Step 3 Accept the move with probability p and exit; or reject the move with probability $1 - p$, go to *Step* 4.

Step 4 Remove the move from the candidate list. If the list is now empty, accept the first move of the original candidate list and exit. Otherwise, go to *Step* 2.

In practice, if the candidate list is moderately large, the above procedure can be simplified by considering a reduced number of moves for probabilistic selection. For that, a pool is created to store a certain number of best moves from the candidate list (penalizing tabu moves as before), thus effectively creating a new and smaller candidate list. This simplification is based on the high probability of choosing one of the first d moves, for modest values of p, even if d is relatively small. Note that the probability of choosing one of the d best moves in the candidate list is $1 - (1 - p)^d$.

Thus if $p = 0.3$, the probability is about 0.832 for picking one of the top five moves, and about 0.972 for picking one of the top ten moves. We selected $p = 0.3$ as a basis for our subsequent experiments.

Instead of using the static value of selection probability p in *Step* 3, we introduce a modification to take fuller account of the relative move evaluations. Specifically, we fine-tune the probability of selection based on the ratio of the move evaluation currently examined to the value of the best solution found so far. This selection probability is calculated by $p^{\alpha r - \beta}$ where r represents the indicated ratio and α and β are positive parameters. With the values of α and β set appropriately, the new probability function provides a fine-tuned probability to discriminate among different evaluations, and favor those proportionately closer to the best solution value. This increases the chance of selecting "good" moves. For example, if α is set to 1.0 and β is set to 0.15, then a move with an evaluation 1.01 times the best solution cost ($r = 1.01$) has a selection probability of 0.355, which is higher than the base probability 0.3; for a move with $r = 1.2$, the selection probability is 0.282, which is lower than the base probability 0.3. In particular, the additional fine-tuned mechanism yields probabilities greater than p for $r \leq (1 + \beta)/\alpha$, and probabilities less than p for $r > (1 + \beta)/\alpha$.

3.6. Advanced Recovery

The use of advanced recovery strategies as an intensification component in Tabu Search has proved effective in a number of applications (see Glover 1996). In this application, we employ a variant proposed in Xu et al. (1996a, 1996b) and Xu and Kelly (1996) that postpones the recovery of elite solutions until the last stage of the search. Each recovered solution launchs a search that constitutes a fixed number of iterations before selecting the next solution to recover. The same elite solution list maintained for error correcting by the advanced evaluator, described in §3.3, serves naturally as a pool of solution for this final stage. Solutions are recovered from this pool in *reverse order*, that is, by starting from the solution with the worst evaluation and working toward the solution with the best evaluation. The list is updated each time a solution is found better than the current worst solution in this elite pool. We merely insert the new solution in its proper location, dropping the worst solution. To enable more elite solutions to be recovered, we thus allow the number of solutions recovered to be larger than the size of the original size of the elite pool. We implement the elite pool for advanced recovery as a circular list, that is, when the best solution (last element) in this pool is recovered, we move back around to the current worst solution (first element) and work toward the best solution again. For each

solution recovered, all tabu restrictions are overridden and reinitialized.

4. Computational Results

In this section, we first report our computational outcomes for two sets of test problems. The problems are generated randomly from distributions whose parameters are selected to create the most difficult problem instances for randomly generated problems from a computational standpoint. The locations of target nodes and steiner nodes are randomly generated in Euclidean space with coordinates from the interval [0, 1000]. Euclidean distances are used for calculating the link costs. The fixed cost of selecting a steiner node is generated randomly from the interval [0, 1000]. We observed that a small fixed cost in this case tends to produce difficult instances because of the "steiner" nature of the problem. The first set of test problems is taken from Lee et al. (1996a), and is restricted to problems of relatively small dimensions that were capable of being solved by the branch and cut approach of their study. Problems from the second test set have larger dimensions, and are beyond the ability of current exact methods to solve. The tables that report our results represent the problem dimensions by `m` and `n`, which identify the number of target and steiner nodes respectively.

We conducted all our tests on a Sun Sparc workstation 20, Model 512 and report CPU time in seconds.

4.1. Parameter Description

An initial solution for our TS approach is produced by linking every target node to its closest steiner node, and then constructing a TSP tour using 2-opt on the set of selected steiner nodes. Since this initial solution does not address the tradeoff between steiner node costs and link costs, it is usually a very poor quality solution. Our TS approach starts from this solution to search for progressively better solutions.

Tabu Tenures for the three types of moves in the TS procedure are randomly generated from an associated (relatively small) interval each time a move is executed. The interval [1, 3] is used for constructive moves and the interval [2, 5] is used for destructive moves. In the case of swap moves, an interval of [1, 3] is used for each of the two elementary moves composing the swap. Most TS applications use intervals that are centered around somewhat larger values. Apparently, the ability to use these small intervals successfully, without cycling, is aided by the oscillation strategy whereby the search alternates between the different types of moves. The smaller Tabu Tenures conceivably help the search explore promising regions more thoroughly under these conditions.

Swap moves are executed either once every seven iterations or in a block of five consecutive iterations when no "new best" solution is found during the most recent 100 iterations. The candidate list for swap moves consists of the top (up to) ten best destructive moves and top (up to) ten constructive moves from the last iteration. At each iteration, the intermediate evaluator is always applied to the top ten best candidate moves estimated by the basic evaluator. The error correction procedure (by the advanced evaluator) is executed each time a "new best" solution is found, and is applied to the current solution after every three accumulated moves, not counting destructive moves that drop nodes of degree one. Error correction is also applied every 100 iterations to the priority queue that stores the 30 best solutions. Also, as mentioned in §3.3, the error correction is not executed before iteration 200 and is omitted when the current TSP tour contains less than ten nodes. The embedded TS-TSP procedure is terminated at 200 iterations. The maximum allowable number of iterations for our complete method is set to 150 for the first test set (which we found to be trivially easy for our method) and 5000 for the second set.

Long-term memory is activated after 500 iterations, so that it can be based on relatively reliable frequency information. The penalty term based on long term memory is calculated by multiplying 320 by the normalized frequency for elementary moves, and multiplying 135 by the sum of the two respective normalized frequencies for swap moves. In probabilistic move selection, we choose the probability of acceptance $p = 0.3$, as previously noted. The parameters for fine-tuned probability described in §3.4 are set as: $\alpha = 1.0$ and $\beta = 0.15$. We additionally use the simplification of shrinking the candidate list for the probabilistic rule to contain the ten best moves ($d = 10$), since

the probability of selecting a move outside the reduced list would be less than 0.03.

Note that all the above parameters are selected intuitively or based on several preliminary experiments, without any attempt at fine tuning. An effort to fine-tune these parameters, for example, using a systematic procedure based on statistical tests (see Xu et al. 1998), may significantly improve the performance of our algorithm.

4.2. Test Results

The first set consists of 175 test problems where m ranges from 10 to 90, n ranges from 10 to 50, and m + n does not exceed 100. For each problem size, we generate five instances using different seeds for random number generator. We report average results for these five instances.

For comparison, we also list the average results for the branch and cut algorithm described in Lee et al. (1996a). We also include solution information for a special heuristic (denoted LS) that is described in Lee et al. (1996a) and provides the upper bound for their exact algorithm. This heuristic strategically generates a set of initial solutions and then improves them using local search. We enclose the description of this heuristic in the appendix. In addition, the LS approach can be significantly enhanced by iteratively restarting the process. That is, at each iteration, we randomly generate an initial solutions and then apply the LS to improve it. The best solution found in all iterations is reported. Since in this restarting extension, the move selections are probabilistically selected based solely on the LS choice criteria, it can be classified as a memoryless variant of probabilistic Tabu Search (see Glover 1996). We denote this latter method by LS-PTS where the number of iterations for restarting is set to 150. Since the exact method based on the mathematical formulation in §2 requires at least three nodes for the TSP tour, we disallow any heuristic solution with less than three active nodes for an equitable comparison.

Since problems of the first test set are relatively small and easy for our algorithm, we reduced the maximum number of iterations to 150. With this stopping criterion, a few advanced features in our TS algorithm, such as the long term memory strategy, and the elite solution error correction and recovery strategies, are disabled. In Table 1, we report the percentage of the error relative to the optimum objective values obtained by the exact method and CPU times of our TS, LS, and LS-PTS methods. In the last column, we list the CPU time required by the branch and cut method on the same machine. Recall that all results are the average values over five instances for the same problem size, and all CPU times are measured in seconds.

Table 1 Computational Results on Small Size Random Problems

Problem ($m \times n$)	TS		LS		LS-PTS		Exact Method
	Error	CPU	Error	CPU	Error	CPU	CPU
(10 × 10)	0	0	11.82	0	0.08	0	1.4
(10 × 20)	0	0	8.72	0	0	0	12.0
(10 × 30)	0	0	4.38	0	0.28	0	45.0
(10 × 40)	0	0	3.09	0	0	0	164.2
(10 × 50)	0	0	9.83	0	0	0	292.4
(20 × 10)	0	0	4.20	0	0	0	2.2
(20 × 20)	0	0	2.96	0	0	0	13.8
(20 × 30)	0	0	1.01	0	0	0	57.4
(20 × 40)	0	0	2.39	0	0	0	287.8
(20 × 50)	0	0	0.82	0	0	0	854.8
(30 × 10)	0	0	0.70	0	0	0	2.8
(30 × 20)	0	0	2.60	0	0	0	21.6
(30 × 30)	0	0	3.79	0	0	0	224.0
(30 × 40)	0	0	2.91	0	0	0.6	351.8
(30 × 50)	0	0	1.08	0	0	1	411.6
(40 × 10)	0	0	0.66	0	0	0	2.6
(40 × 20)	0	0	2.39	0	0	0	22.6
(40 × 30)	0	0	2.80	0	0.06	0.8	80.2
(40 × 40)	0	0	3.71	0	0	1.0	719.8
(40 × 50)	0	0	1.79	0	0.03	1.0	1037.6
(50 × 10)	0	0	0.43	0	0	0	3.6
(50 × 20)	0	0	1.72	0	0	0.4	37.8
(50 × 30)	0	0	2.48	0	0.05	1.0	139.6
(50 × 40)	0	0	2.38	0	0	1.0	384.8
(50 × 50)	0	1	1.20	0	0	2.4	1003.8
(60 × 10)	0	0	1.15	0	0	0	4.6
(60 × 20)	0	0	0.89	0	0	1	30.2
(60 × 30)	0	0.2	0.01	0	0	1.2	108.6
(60 × 40)	0	0.8	1.37	0	0.04	2.4	368.6
(70 × 10)	0	0	0.80	0	0	0	5.0
(70 × 20)	0	0	2.35	0	0	1.0	35.6
(70 × 30)	0	0.2	1.23	0	0	2.0	175.0
(80 × 10)	0	0	0.55	0	0	0	6.0
(80 × 20)	0	0	1.60	0	0.20	1.4	34.4
(90 × 10)	0	0	0.81	0	0	0.4	5.8

From Table 1, we find that the computation times for the exact method increase exponentially with n for each fixed m. Consequently, it is truly hard to solve the larger instances of the ring-based DDS problem using the current exact method. The LS is very fast and obtains good solutions, but it cannot find the optimal solutions for all five problem instances for any problem size. LS-PTS significantly improves LS at very reasonable extra computational effort, finding optimal solutions for the five problem instances in 28 out of the 35 different problem sizes tested (hence in 80% of these problem sizes). Our TS performs extremely well by finding optimal solutions for all problem instances in all problem sizes (hence for all 175 test problems). We emphasize that the TS procedure we are testing in these cases is a simple TS algorithm without the assistance of advanced features.

We then extended our tests to larger problem instances. The dimensions for the second set of test problems are as follows. The value of n for the first 15 problems ranges from 100 to 200 in increments of 25. For each n, three problems are generated by setting m equal to n, $n + 50$ and $n + 100$ respectively. The last six problems in this set are designed to be particularly large and have dimensions 250×250, 300×250, 350×250, 100×300, 200×300, and 300×300. Since exact methods are unable to handle problems of this second set and it is also difficult to find a reasonably good lower bound from the mathematical formulation, we evaluate the TS heuristic by comparing its performance to those of the LS and LS-PTS heuristics, which proved capable of finding optimal or near-optimal solutions for the first set of problems. Since the problems are large, the search termination condition is extended to 5000 iterations, which enables the advanced features of our TS algorithm.

As for the first problem set, we generate five instances for each problem size in the second problem set. Since our Tabu Search algorithm outperforms the LS and LS-PTS, we report the outcomes in the form of error percentages of the LS and LS-PTS over TS. We list the maximum (MAX), minimum (MIN), and average (AVG) error percentages for each problem size in Table 2. In addition, we also list the number (NUM) of problems where TS improves the LS or LS-PTS among the five instances.

From Table 2, we observe that TS consistently outperforms LS and LS-PTS. In particular, TS improves LS solutions in 104 instances out of 105 test problems with average cost savings of 3.64%. Compare with the solutions obtained by LS-PTS, TS improves 79 LS-PTS solutions and the average improvement is 0.23%. The magnitude of improvement is more noticeable for larger problem size. Given the relatively good performance of LS-PTS in the first problem set (where 80% of the LS-PTS solutions are in fact optimal solutions), the improvement by our TS method on this larger test set is quite significant. Such an improvement provides a valuable competitive edge in attracting customers, with the associated benefit of increasing the company's market share and profits.

We also compare the CPU time required by TS with those required by LS and LS-PTS. We list the maximum (MAX), minimum (MIN), and average (AVG) CPU time (in second) by each algorithm among the five instances for each problem size in Table 3. The times reported herein are times required by obtaining the best solutions for the corresponding heuristic.

Table 3 discloses that the TS uses very reasonable CPU times to obtain high quality solutions and can meet the time requirement for real-world applications. LS uses much less CPU time, however, it can be easily improved by LS-PTS and TS. The enhanced local search method, LS-PTS, though taking advantages of using more CPU time and randomly escaping local optima, is still outperformed by our TS heuristic. This confirms the more "intelligent" nature of Tabu Search over the local search techniques.

Finally we present an algorithmic analysis to investigate the relative contributions made by the various components of our TS algorithm. We test a series of variants which disable certain TS components on the second problem set. The variants under investigation include the one without short-term memory (STM), the one without long-term memory (LTM), the one without probabilistic selection rule (PSR), the one without advanced recovery strategy (ARS), the one without the use of advanced evaluator (AE), and the simple TS (STS) which we tested in the first set of

Table 2 Cost Comparisions on Larger Size Random Problems

	LS over TS				LS-PTS over TS			
Problem ($m \times n$)	MAX	MIN	AVG	NUM	MAX	MIN	AVG	NUM
(100 × 100)	4.77	1.89	2.93	5	0.33	0	0.07	1
(150 × 100)	3.42	0.8	1.94	5	0.5	0	0.2	3
(200 × 100)	3.32	0.78	2.06	5	0.48	0	0.17	4
(125 × 125)	3.93	1.15	2.28	5	0.39	0	0.2	4
(175 × 125)	3.03	0.36	1.17	5	0.3	0	0.1	3
(225 × 125)	3.94	0.08	2.35	5	0.6	0	0.2	4
(150 × 150)	7.02	1.5	4.58	5	0.36	0	0.13	2
(200 × 150)	2.94	0.26	1.49	5	0.38	0	0.11	3
(250 × 150)	4.01	1.31	2.96	5	0.58	0	0.24	3
(175 × 175)	2.37	0.83	1.46	5	0.38	0	0.21	4
(225 × 175)	4.31	0.98	2.54	5	0.25	0	0.12	4
(275 × 175)	2.99	1.16	2.02	5	0.22	0	0.06	3
(200 × 200)	2.31	0	1.04	4	0.56	0	0.27	3
(250 × 200)	3.25	0.51	1.71	5	0.2	0	0.12	4
(300 × 200)	2.73	1.91	2.2	5	0.44	0.03	0.29	5
(250 × 250)	3.47	2.07	2.69	5	0.59	0.13	0.32	5
(300 × 250)	3.15	2.04	2.56	5	0.7	0.39	0.57	5
(350 × 250)	2.37	0.75	1.48	5	0.5	0.05	0.29	5
(100 × 300)	4.53	0.61	2.88	5	0.83	0	0.28	4
(200 × 300)	3.64	2.15	2.73	5	0.98	0.12	0.39	5
(300 × 300)	4.94	2.27	3.41	5	0.7	0.11	0.39	5
Average	3.64	1.11	2.31	4.95	0.49	0.04	0.23	3.76

problem (which disables the LTM, ARS, AE, and terminates at 150 iterations). For ease of exposition and to simplify the comparisons, we only report the percentage of problems in which the variant could not match the best TS solutions reported in Table 2. In other words, this percentage indicates the degree of improvement that the corresponding component can contribute. The comparisons are presented in Table 4.

The outcomes from Table 4 validate that all components can significantly enhance the basic Tabu Search algorithm. In particular, the long-term memory and the probabilistic selection rule play important roles in diversification and therefore improve the overall search quality. The advanced recovery strategy provides an effective intensification role and helps locate better solutions in late stages of the search. The short-term memory is primarily used to prevent the search from revisiting local optima and to reduce the chance of cycling. Though these functions are diminished by the introduction of the probabilistic move selection strategy, the short-term memory still make a notable impact. The use of probabilistic move selection additionally affords an effective means to compensate for the noise caused by the approximate move evaluation. Incidentally, we note that the advanced evaluator (incorporating TS-TSP) is impressive in finding better TSP tours than 2-opt, and it can be efficiently executed within a hierarchical framework. The impact of integrating the more advanced TS components such as LTM, PSR, ARS and AE becomes even more evident since they improve the solutions in nearly 70% of the problem instances, by comparison with the elementary TS heuristic (which in this application embraces probabilistic move choice and short-term memory).

5. Conclusion

We have developed and tested alternative Tabu Search implementations for solving a ring-based

Table 3 CPU Time Comparisions on Larger Size Random Problems

	LS			LS-PTS			TS		
Problem ($m \times n$)	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG
(100 × 100)	0	0	0	31	31	31	4	1	2.6
(150 × 100)	0	0	0	62	61	61.6	85	2	26.4
(200 × 100)	0	0	0	104	101	102.4	9	2	5
(125 × 125)	1	0	0.2	58	57	57.4	36	2	12.2
(175 × 125)	1	1	1	104	102	103	51	3	20.2
(225 × 125)	1	1	1	164	158	160.2	143	4	37.8
(150 × 150)	1	1	1	96	95	95.6	133	3	40
(200 × 150)	1	1	1	163	158	159.8	95	5	42
(250 × 150)	1	1	1	250	238	242	115	29	74.6
(175 × 175)	1	1	1	148	146	146.8	85	8	42.8
(225 × 175)	2	1	1.8	230	227	228.8	118	11	55.4
(275 × 175)	2	2	2	334	332	332.6	307	44	161.6
(200 × 200)	2	2	2	219	214	216.4	23	6	10.6
(250 × 200)	2	2	2	331	324	326.8	387	12	164.2
(300 × 200)	2	2	2	473	461	465.6	178	15	63.2
(250 × 250)	4	3	3.6	431	420	424.4	448	120	249.8
(300 × 250)	4	3	3.2	598	585	588.6	420	21	254.4
(350 × 250)	5	4	4.2	771	763	767.4	537	61	301
(100 × 300)	1	1	1	98	96	96.4	116	4	46
(200 × 300)	3	2	2.2	344	338	341	280	17	209
(300 × 300)	8	5	6.2	729	709	719	649	23	356

DDS network design problem encountered in telecommunications industry. In our approach, the search incorporates constructive and destructive moves as well as exchange moves to explore different neighborhood structures. We introduce evaluation estimates to allow moves to be selected more efficiently, and accompany these estimates with an error correction procedure that employs hierarchical move evaluators in order to offset the risk of making improper choices. Long-term memory and probabilistic move selection are also included for diversification while the advanced recovery strategy is implemented for intensification.

Table 4 Tests on Various TS Components

Variant	Contribution (%)
STM	12.4
LTM	32.4
PSR	14.3
ARS	32.4
AE	48.6
STS	68.9

Numerical tests, for two sets of randomly generated test problems, show that for the 175 smaller test problems (up to 100 nodes), a simple variant of our TS algorithm yields optimal solutions in all cases while using only a very small fraction of the CPU time required by the exact method (running about three orders of magnitude faster). For the 105 larger problems, the Tabu Search algorithm consistently outperforms the best local search heuristic previously available, including a probabilistic enhancement of this heuristic designed in this study. Our outcomes also demonstrate the relative contributions of short-term memory, long-term memory, probabilistic move selection, advanced recovery, and the advanced move evaluator, showing that the combination of these components can obtain significantly better results than the simple TS version. The gains afforded by the advanced components of tabu

search become more appreciable as the problems increase in complexity.

Future improvements of our TS approach are anticipated to result by including additional long term memory functions and by using more refined candidate list strategies. We observe that some of the steiner nodes always reside in the active set for good solutions, while other are always inactive. An intensification strategy that takes advantage of this fact could yield additional useful information for probabilistic TS designs. In addition, we anticipate that the use of evolutionary strategies, such as scatter search and path relinking (Glover 1977, 1996), may provide an effective post-optimization approach for our TS algorithm.

Appendix

In this appendix, we describe the LS heuristic procedure that has been used to provide an initial upper bound on the optimal solution value in the branch-and-cut algorithm in Lee et al. (1996). The following notation and definitions will be used for that purpose. First recall that m is the number of target nodes and n is the number of steiner nodes. A *star* is a subgraph that consists of a single steiner node (the *center* of the star) and a set of target nodes with edges connecting them to the center. The *weight* of a star is equal to the sum of its edge costs and its steiner node cost. The *size* of a star is equal to the number of target nodes contained in that star. Finally, the *steiner number* and *TSP tour* of a solution are defined respectively as the number of steiner nodes being used and the TSP tour connecting these nodes in that particular solution.

Heuristic Procedure for the Ring-Based DDS Problems

For star size $k = 2, 3, \ldots, m$, repeat the following steps:

Step 1. (Generating an initial current solution)

Step 1.1. Label all nodes in $M \cup N$ "unselected" and set $i = 1$. While $i \leq \min\{\lceil m/k \rceil, n\}$, determine the minimum-weight star of size k that contains only unselected nodes (the last iteration may find a smaller star), and then label all the nodes in the star "selected"; set $i = i + 1$. Each selected target node has been currently assigned to the center of its star.

Step 1.2. Reassign each selected target node in M to its closest selected steiner node in N if necessary.

Step 1.3. If any, assign each unselected target node in M to its closest steiner node in N and then label it "selected."

Step 1.4. Connect all selected steiner nodes in N with a TSP tour using random insertion.

Step 2. If the steiner number of the current solution is greater than or equal to 4, try to improve the solution as follows:

Step 2.1. Improve the TSP tour using 2-opt heuristic.

Step 2.2. Further improve the TSP tour using Or-opt heuristic.

Step 3. If the steiner number of the current solution is greater than or equal to 2, perform the following steps for each selected steiner node:

Step 3.1. Generate a new temporary solution by deleting the selected steiner node from the current solution as follows: remove the selected steiner node from the TSP tour; reassign its target nodes to the closet remaining steiner nodes in the tour; connect the two neighbors of the steiner node in the TSP tour.

Step 3.2. Replace the current solution with the temporary solution if the latter is better.

Step 4. For each unselected steiner node, perform the following steps:

Step 4.1. Generate a new temporary solution by adding the unselected steiner node to the current solution as follows: insert the new steiner node into the TSP tour such that the increase of the tour length is minimized; reassign target nodes to the newly-added steiner node if it is closer.

Step 4.2. Replace the current solution with the temporary solution if the latter is better.

Step 5. If any improvement is made to the current solution in *Step* 3, or 4, go back to *Step* 2.

If the current solution is better than the best solution found, record the current solution as the new best solution found.

References

Anderson, A., K. F. Jones, J. Ryan. 1993. Path assignment for call routing: An application of tabu search. *Ann. Oper. Res.* **41** (J. C. Baltzer), 299–312.

Glover, F. 1977. Heuristics for integer programming using surrogate constraints. *Decision Sci.* **8** 156–166.

——. 1989. Tabu search—Part I. *ORSA J. Computing* **3** 190–206.

——. 1996. Tabu search and adaptive memory programming—Advances, applications and challenges. Barr, Helgason, Kennington, eds. *Interfaces in Computer Science and Operations Research*. Kluwer Academic Publishers, Boston, MA. 1–75.

——, M. Laguna. 1997. *Tabu Search*. Kluwer Academic Publishers, Boston MA.

——, A. Løkketangen. 1996. Probabilistic move selection in tabu search for zero-one mixed integer programming problems. I. H. Osman and J. P. Kelly, eds. *Meta-Heuristics*: *Theory and Applications*. Kluwer Academic Publishers, Boston, MA. 467–487.

Johnson, D. S., L. A. McGeoch. 1996. The traveling salesman problem: A case study in local optimization. E. H. L. Aarts, J. K. Lenstra, eds. *Local Search in Combinatorial Optimization*. John Wiley and Sons, New York.

Laguna, M. 1994. Clustering for the design of SONET rings in interoffice telecommunications. *Management Sci.* **40**(11) 1533–1544.

——, F. Glover. 1993. Bandwidth packing: A tabu search approach. *Management Sci.* **39**(4) 492–500.

Laporte, G. 1992. The traveling salesman problem: An overview of exact and approximate algorithms. *European J. Oper. Res.* **59** 231–247.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys.

1985. *The Traveling Salesman Problem*: *A Guided Tour of Combinatorial Optimization*. Wiley, Chichester, UK.

Lee, Y., S. Y. Chiu, J. Ryan. 1996a. A branch and cut algorithm for a steiner ring-star problem. Working Paper. U S WEST Advanced Technologies Inc., Boulder, CO.

——, ——, ——. 1996b. A branch and cut algorithm for a steiner tree-star problem. *INFORMS J. Computing* **8**(3) 194–201.

Reinelt, G. 1994. *The Traveling Salesman*: *Computational Solutions for TSP Applications*. Springer-Verlag, Heidelberg, Germany.

Skorin-Kapov, D., J. Skorin-Kapov. 1994. On tabu search for the location of interacting hub facilities. *EJOR* **73** 502–509.

Xu, J., S. Y. Chiu, F. Glover. 1996a. Using tabu search to solve the steiner tree-star problem in telecommunications network design. *Telecommunication Systems* **6** 117–125.

——, ——, ——. 1996b. Probabilistic tabu search for telecommunications network design. *Combinatorial Optimization*: *Theory and Practice* Vol. 1, (1) 69–94.

——, ——, ——. 1998. Fine-tuning a tabu search algorithm with statistical tests. *Internat. Trans. Oper. Res.* **5**(3) 233–244.

——, ——, ——. 1997. Tabu search for dynamic routing communications network design. *Telecommunication Systems* **8** 55–77.

——, J. P. Kelly. 1996. A new network flow-based tabu search heuristic for the vehicle routing problem. *Trans. Sci.* **30**(4) 379–393.

Accepted by Thomas M. Liebling; received June 1997. *This paper has been with the authors* 5 *months for* 1 *revision.*