Steiner Trees in Industries
X. Cheng and D.-Z. Du (Eds.) pp. 441 - 466
©2001 Kluwer Academic Publishers

A First Level Scatter Search Implementation for Solving the Steiner Ring Problem in Telecommunications Network Design

Jiefeng Xu
Delta Technology, Inc., 1001 International Boulevard, Atlanta, GA 30354-1801.
E-mail: jiefeng.xu@delta-air.com

Steve Chiu
Genuity Inc., 3 Van de Graaff Drive, Burlington, MA 01803.
E-mail: schiu@genuity.com

Fred Glover
Hearin Center for Enterprise Science, University of Mississippi, University, MS 38677
E-mail: fglover@bus.olemiss.edu

Contents

1 Introduction **442**

2 The Tabu Search Heuristic **445**
2.1 Elementary Tabu Search Procedure 445
2.2 Advanced TS Components 448

3 The SS/PR Algorithm **450**
3.1 General Form of the SS/PR Procedure 450
3.2 Diversification Generators 451
3.3 Improvement Method 453
3.4 Maintaining And Updating The Reference Set 453
3.5 Choosing Subsets of the Reference Solutions 454
3.6 Solution Combination Method 456

4 Computational Results 458
4.1 Parameter Description 459
4.2 Test Results 459

5 Conclusion 462

References
Appendix A

1 Introduction

In this paper, we consider a combinatorial optimization problem in a graph $G(V, E)$ where V represents the node set and E denotes the edge set. V is partitioned into two subsets V_1 and V_2, which denote the "Steiner" node set and "Target" node set respectively. Similarly, E is partitioned into sets of E_1 and E_2, where E_1 is the set of edges that join two Steiner nodes, and E_2 is the set of edges that join one Steiner node and one target node. No edges exist between two target nodes. The problem is to select a non-empty Steiner set $S \subset V_1$ such that each target node must be linked to exactly one selected Steiner node. Meanwhile, all selected (active) Steiner nodes must be connected to form a *Hamiltonian* tour of minimum connection cost which is called *Traveling Salesman (TSP)* tour. The objective of the problem is to minimize the total costs (which we will describe in detail subsequently).

This problem can be viewed as a variant of the Steiner Tree family. (See for example the survey by Chopra and Rao [1] and Duin and Voβ [2].) Our problem is an extension of a simpler "predecessor" problem which stipulates that all active Steiner nodes must form a minimum spanning tree. The predecessor problem can be formulated as a degree constrained Steiner tree problem with both node and edge costs, where the target nodes form a star topology around the active Steiner nodes, and carries the name Steiner Tree Star (STS) problem. Similarly, we call our problem the Steiner Ring Star (SRS) problem.

The STS and SRS problems are critical in designing Digital Data Service (DDS) networks in the telecommunications industry. These networks use permanent connections and dedicated transmission facilities to provide high quality services. In addition, the ring-based SRS model prevails since it provides better reliability than the tree-based STS model. In the DDS application, we have input elements which include a set of end offices, a set of digital hubs and a set of customer locations that are geographically distributed on a plane. The target nodes consist of the customer locations while the Steiner nodes represent the digital hubs. 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. Each hub has a fixed cost for being chosen and each link has a connection cost for being included in the solution.

The connection cost between the two selected digital hubs, and the connection cost between the customer location to a selected hub (via its uniquely designated end office), are distance sensitive and can be pre-calculated according to the tariff charges established by Federal Communications Committee (FCC). (See Xu, Chiu and Glover [19] for more detail on calculating these costs.) The FCC requests the telecommunications companies to provide the least-cost network design to customers, and the SRS model can be used to provide such a network design at minimum cost.

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. Telecommunications companies require an automatic quoting system that allows the sales representative to give the customer a quote based on the SRS model over the phone within one minute. A significant 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.

The STS and SRS problems have received some attention from operations researchers recently. Lee et al. [12] show that the STS problem is strongly NP-hard and identify two mixed zero-one integer programming formulations for this problem. Lee, Chiu and Ryan [13] further propose a branch and cut algorithm for the STS problem. Xu, Chiu and Glover ([15], [16]) conduct a series of studies on solving the STS problem using an advanced Tabu Search (TS) heuristic. More recently, Xu, Chiu and Glover [20] extend their TS studies by exploring the use of Scatter Search (SS) for the STS problem, which they find to be competitive with the advanced tabu search algorithm in terms of solution quality and times. For the SRS problem, Lee, Chiu and Ryan [14] again propose a branch and cut algorithm similar to the one for the STS problem, and Xu, Chiu and Glover [19] provide a comprehensive research on developing an advanced TS heuristic. Numerical tests reported show that for the 175 smaller test problems (up to 100 nodes), a simple variant of the TS algorithm yields optimal solutions in all cases while using only a very small fraction of the CPU time required by the exact method proposed by Lee, Chiu and Ryan (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.

This paper extends our research in Xu, Chiu and Glover ([19], [20]) by exploring an implementation of Scatter Search (SS) for the SRS problem. Scatter search and its generalized form called path relinking (PR) are evolutionary methods that have recently been shown to yield promising outcomes for solving combinatorial and nonlinear optimization problems including the STS problem. Based on formulations originally proposed in the 1960s ([3], [4]) for combining decision rules and problem constraints, the methods use strategies for combining solution vectors that have proved effective for scheduling, routing, financial product design, neural network training, optimizing simulation and a variety of other problem areas (see, e.g., the survey of [8]).

Our goal is to determine if it is possible to create a simple implementation of SS that can compete with the best approach previously devised. The previously best method, based on tabu search, is the outgrowth of a lengthy and intensive development and testing process. One of the useful features of scatter search is that it is highly compatible with tabu search, and in fact arises from common underlying principles, by which scatter search can be viewed as a method to achieve a certain integration of intensification and diversification (in tabu search terminology).

Consequently, if a straightforward implementation of scatter search can be documented to provide a strong performance in relation to tabu search, the outcome will motivate future research to integrate these two approaches in a more sophisticated procedure. We do not undertake to fully detail the elements of such a more advanced procedure, since there are a number of possible variations. Rather, we seek to establish whether the effort to create such an advanced method may be warranted, and to provide an indication of the fundamental character of the scatter search component of such a procedure.

In the following development, we will not only describe our SS approach, but also give background details of the tabu search method that has produced the best performance to date in solving these problems, as a basis for clarifying elements of compatibility between the approaches and for understanding the alternatives that exist for creating a higher level integration of the procedures.

This paper is organized as follows. We first describe the tabu search algorithm for the SRS problem in Section 2. We further describe the SS/PR based heuristic for the SRS problem in section 3 and examine several relevant issues, such as the diversification generator, the reference set update method, the subset generation method, the solution combination method and the improvement method. In section 4, we report computational results on a set of carefully designed test problems, accompanied by comparisons with the solutions obtained by the TS algorithm [19] which has been documented as the best heuristic available prior to this research. In the concluding section, we summarize our methodology and findings. To further improve the readability, we present the mathematical formulation of the SRS problem in Appendix A.

2 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. The TS framework can also be used to guide choices made in successive passes of multi-start methods, or to control the process of selecting neighborhood moves for multiple neighborhood methods. In the subsequent subsections, we briefly describe the tabu search algorithm first proposed in Xu, Chiu and Glover [19]. Our purpose is to provide a basic knowledge of a form of that has worked well in the present setting, and to disclose similarities and differences between our TS method and the scatter search/path relinking method that is the focus of our current investigation. The paper by Xu, Chiu and Glover [19] provides further details of our TS method for the SRS problem, and the survey by Glover and Laguna [9] provides a complete review of TS.

2.1 Elementary Tabu Search Procedure

Tabu search is an iterative search method which can be used to guide traditional local search methods to escape the trap of local optimality. At each iteration, a set of candidate moves is extracted from the neighborhood for evaluation, and a "best" (highest evaluation) move is selected, thereby generating a new solution. During each iteration, certain neighborhood moves are considered *tabu* and excluded from the candidate list. A best non-tabu move can be identified either by deterministic or probabilistic selection mechanisms. Aspiration criteria are typically introduced that can override the tabu status of move to allow a tabu move to be selected if it is sufficiently attractive by these criteria. The algorithm proceeds until a pre-defined number of iterations elapses and then terminates to output the all-time best solution found.

Neighborhood Structure and Moves

We make use of the following two elementary types of neighborhood moves for the SRS problem: a constructive move which changes an inactive Steiner node to an active one (by inserting the node into the current TSP tour); a destructive move that changes an active Steiner node to an inactive one (by deleting the node from the current TSP tour). We also use the pairwise exchange (swap) moves, which exchange one active Steiner node with one inactive Steiner node. A swap move can be viewed as a combination of a constructive and a destructive move. It introduces a more significant change to the current TSP tour, and possesses a larger neighborhood space than the constructive and destructive moves. To reduce the number of swap moves to be evaluated, we construct a candidate list to restrict attention to pairs consisting of (up to) ten best destructive and (up to) ten best constructive moves, considered in isolation.

We blend these three different types of moves to produce the complete neighborhood search. Swap moves are performed more sparingly, taking the roles of producing periodic perturbation and conditional oscillation. In this application, 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. This approach of cyclying among multiple neighborhoods, invoking the more complex neighborhoods less frequently or when progress in simpler neighborhoods slow down, is one of the early forms of strategic oscillation used in tabu search (see, e.g., Glover and McMillan [10]).

Tabu Search Memory

Short Term Tabu Search Memory. Elementary moves are classified *tabu* as follows: if the node x is currently dropped from the active Steiner node set, we forbid this node to move back to the active set for a prescribed number of iterations. Similarly, if the node x is currently added to the active Steiner node set, we forbid this node to be dropped by moving to the inactive set for a specified number of iterations. For swap moves, we impose these 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 the active set and moving node y back to the inactive set.

We determine the duration (called the tabu tenure) over which the tabu status of a move remains in effect by sampling uniformly from a predetermined interval at each iteration. We use the simple aspiration criterion that overrides the *tabu* restriction if the current candidate move would lead to a new best solution.

Short term memory to exploit these restrictions is implemented using a recency based memory structure as follows. Let iter denote the current iteration number and let $\mathsf{tabu\_add(x)}$ and $\mathsf{tabu\_drop(y)}$ 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). Let $\mathsf{U(a,b)}$ be the tabu tenure which is uniformly generated from the interval $[a,b]$. Initially, $\mathsf{tabu\_add(x)}$ and $\mathsf{tabu\_drop(x)}$ 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:

$$\mathsf{tabu\_add(x)} = \mathsf{iter} + \mathsf{U}(1,3) \text{ (for the constructive move of adding node x)},$$
$$\mathsf{tabu\_drop(y)} = \mathsf{iter} + \mathsf{U}(2,5) \text{ (for the destructive move of dropping node y)}.$$

Then the restriction to prevent x from being dropped is enforced when $\mathsf{tabu\_add(x)} > \mathsf{iter}$, and the restriction to prevent y from being added is enforced when $\mathsf{tabu\_drop(y)} > \mathsf{iter}$. For the swap move, we impose this restriction by comparing *iter* with both *tabu_add* and *tabu_drop* memory. At each iteration, we select the move with the highest move evaluation (lowest cost), requiring the move to be non-tabu unless the evaluation is high enough to permit the aspiration criteria to apply. Details of the move selection procedure incorporating the TS restrictions and aspiration criterion, including the pseudo code, are available in Xu, Chiu and Glover [19].

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. We use a transition measure to record the number of times each Steiner node changes from an active status to an inactive status or vice versa. This measure is normalized to lie in the interval [0,1] and then is linearly scaled by a selected constant to create a penalty term, which is added to the corresponding move evaluation. In this application, the penalty term 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. The long term memory is activated after 500 iterations to allow the frequency information to be more reliable.

2.2 Advanced TS Components

Several advanced TS components were developed for the SRS problem which significantly enhanced the overall performance of the method.

Hierarchical Move Evaluation

For a neighborhood move where the active Steiner node set is determined, it is trivial to calculate the connection cost for assigning each target node to its nearest active Steiner node, and to calculate the sum of all (active Steiner) node costs. Thus the core of the move evaluation is to find an optimal TSP tour over the active Steiner nodes. We devised a hierarchical evaluation mechanism which employs the heuristic evaluators at three different levels (basic, intermediate, advanced) as follows.

Basic Evaluator. The basic evaluator is used to evaluate each constructive, destructive and swap move in the candidate list. For constructive moves, the evaluator identifies the minimum insertion cost, i.e. the cost that results 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 a standard 2-opt heuristic to improve the current tour. The 2-opt progressively improves the tour by considering possible ways of removing two arcs from the current tour and then reconnecting the two resulting chains to form a new TSP tour, terminating when no improvement can be obtained. The 2-opt procedure in this application can be significantly simplified with our destructive and constructive moves. Suppose the current tour is already locally optimal. Then we need to evaluate only the options that remove new edges that the destructive move and the constructive move introduce to the tour (one new edge in the case of destructive moves, two new edges in the case of constructive moves). For swap moves, which change the tour in a more complex way, we simply apply the standard 2-opt procedure. In this application, we apply the intermediate evaluator to the top ten best candidate moves identified by the basic evaluator at each iteration.

Advanced Evaluator The advanced evaluator uses more complicated search techniques for improvement. First, it applies 3-opt local search to improve the current tour. Then it employs a stand alone simple TS algorithm for the TSP (TS-TSP), which uses simple node ejection and swap moves, and a rudimentary short term memory structure to search for solutions beyond local optimality. The TS-TSP method was first successfully used as a tour-improvement tool in the Vehicle Routing Problem (VRP) by Xu and Kelly [21]. Computational experience showed that the TS-TSP approach often produced shorter TSP tours than 3-opt for moderate or large sized problems. Therefore, we execute this evaluator to correct the costs estimated by the two simpler evaluators. We run the advanced evaluator in this application whenever: (1) a "new best" solution is found; (2) the current solution accumulates three moves whose costs were estimated by the intermediate evaluator; (3) an "elite solution" has been on the list of the top thirty best solutions for 100 iterations and has not yet been processed by the advanced evaluator to correct its estimation errors. The advanced evaluator is not activated before iteration 200, and is not executed if the current tour contains less than ten nodes.

Probabilistic Move Selection

We apply probabilistic tabu search [6] to combat "noise" caused by cost estimation in the move evaluation. To do this, we evaluate all moves from the candidate lists and rank them by their estimated costs. (*Tabu* moves are penalized to received high costs unless the aspiration criterion applies.) In the selection phase, we first examine the move with the lowest cost. If the move satisfies the aspiration criterion, it is automatically selected. Otherwise, we select the move with the probability $1 - p$. If the current move is rejected, we examine the move with the next lowest cost and repeat the process until a move is selected.

The probability of choosing one of the d best moves in the candidate list is $1 - (1 - \mathsf{p})^{\mathsf{d}}$. In practice, we set $d = 10$ by keeping the top 10 best move evaluations for selection. In the rare case where all 10 moves are rejected, we simply select the move with the lowest cost by default. We also fine-tune the value of p as $0.3^{r-0.15}$ where r is the ratio of the move evaluation currently examined to the value of the best solution found so far. This allows "good" moves an increased chance to be selected.

Advanced Solution Recovery Strategy

We apply an advanced solution recovery strategy for intensification purposes. In this application, we start recovery of elite solutions at iteration 1800. Each recovered solution launches a search that continues for 80 iterations before selecting the next solution to recover. Solutions are recovered from the elite list (consisting of the top 30 best solutions found so far, sorted by their cost estimates) 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 dynamically whenever a solution is found that is better than the worst solution in the list. Then the new solution is inserted into the proper position in the list and the worst solution is dropped. The elite list for advanced recovery is implemented as a circular list, that is, when the last solution in this list is recovered, we move back around to the first (current worst) solution and work toward the best solution again. For each solution recovered, all previous tabu restrictions are dropped and the search begins again from the solution with a "pure slate".

3 The SS/PR Algorithm

In the following subsections, we first describe the general form of our SS/PR algorithm, and then describe each component which is specifically designed for the SRS problem.

3.1 General Form of the SS/PR Procedure

In general, SS/PR methods consist of the following components:
(1) A **Diversification Generator**: to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.
(2) An **Improvement Method**: to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the "enhanced" solution is considered to be the same as the input solution.)
(3) A **Reference Set Update Method**: to build and maintain a Reference

Set consisting of the b best solutions found (where the value of b is typically small, e.g., between 20 and 40), organized to provide efficient accessing by other parts of the method.
(4)A **Subset Generation Method**: to operate on the Reference Set, to produce a subset of its solutions as a basis for creating combined solutions.
(5) A **Solution Combination Method**: to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solution vectors.

All aforementioned components come from a standard template used in scatter search and path relinking. The framework consists of two phases: an initial phase and the SS/PR phase. In the initial phase, we create one or more initial solutions, then use the **Diversification Generator** to generate diverse trial solutions from the seed solution(s). For each trial solution produced, we use the **Improvement Method** to create one or more enhanced trial solutions. During this procedure, we maintain and update a **Reference Set** consisting of the b best solutions found. We repeatedly execute this procedure until producing some designated total number of enhanced trial solutions as a source of candidates for the *Reference Set*.

The SS/PR Phase proceeds iteratively as follows. At each iteration, we use the **Subset Generation Method** to generate subsets of the **Reference Set** as a basis for creating combined solutions. For each subset X produced, we apply the **Solution Combination Method** to produce a set of new combined solutions $C(X)$. For each solution in C(X), we improve it using the **Improvement Method** while continuing to maintain and update the Reference Set. We repeat the procedure until the termination conditions are met.

We describe in detail each of the components of the foregoing template in the subsequent subsections.

3.2 Diversification Generators

As implied before, an SRS solution can be conveniently represented by an 0-1 vector of the decision variables which determine if the corresponding Steiner node is active or not. We supply a seed vector $X(0) = x_1, ..., x_n$, from which a set of diversified solution vectors is produced using the following *Diversification Generator for Zero-One Solutions.* . Let $NumSolutions$ represent the number of unique solution vectors currently collected, and $MaxNumSolutions$ indicate the maximum number of unique solutions required for the Reference Set. $h*$ and $q*$ are two integer parameters require for the algorithm. Then the Diversification Generator proceeds as follows.

```
$NumSolutions = 0$
$X(NumSolution) = x_1, \ldots, x_n$
for $h = 1$ to $h*$
    Let $q* = 1$ if $h < 3$, and otherwise let $q* = h$
    for $q = 1$ to $q*$
        let $k* = \lfloor (n - q)/h \rfloor$
        let $x'$ and $x''$ be two zero $n$-vectors. Set $x'_1 = 1$.
        for $k = 1$ to $k*$
            $x'_{q+kh} = 1 - x_{q+kh}$
        end k
        If $h > 1$ then
            $x''_i = 1 - x'_i \quad \forall i = 1, \ldots, n$
            $X(NumSolution + 1) = x'_1, \ldots, x'_n$
            $X(NumSolution + 2) = x''_1, \ldots, x''_n$
            $NumSolutions = NumSolutions + 2$
        else
            $X(NumSolution + 1) = x'_1, \ldots, x'_n$
            $NumSolutions = NumSolutions + 1$
        end if
        If $NumSolutions >= MaxSolutions$, then stop generating
            solutions.
    end $q$
end $h$
```

In the above algorithm, if we fix $q* = 1$, the diversification generator first produces solution vectors associated with an integer $h = 1, 2, \ldots, h*$, where $h* < n - 1$. We recommend that $h* < n/5$ since as h becomes larger, the solutions for two adjacent values of h differ from each other proportionately less than when h is smaller. Then the integer $q = 1, \ldots, q*$ shifts the resulting solution vectors to the right by adding q leading zeros. This creates more diversified solution vectors.

The number of solutions x' and x'' produced by the preceding generator is approximately $q * (q * +1)$. Thus if $n = 50$ and $h* = n/5 = 10$, the method will generate about 110 different output solutions, while if $n = 100$ and $h* = n/5 = 20$, the method will generate about 420 different output solutions. To prevent the number of output solutions from growing too fast as n increases, while creating a more diverse subset of solutions, we can skip over various q values between 1 and $q*$. The greater the number of values skipped, the less "similar" the successive solutions (for a given h) will be. Similarly, h itself can be incremented by a value that differs from 1. In our implementation, we set MaxSolutions equal to the number of "empty slots" in the reference set, so the procedure terminates either once the reference set is full, or after all of the indicated solutions are produced.

3.3 Improvement Method

We apply a local search heuristic to improve any initial solution or trial solution fed into the **Improvement Method**. The trial solutions include those produced by the diversification generator and the combination method. The local search heuristic is an iterative method which employs the same neighborhoods of moves used for the tabu search algorithm, i.e., constructive moves, destructive moves and swap moves. At each iteration, we in turn evaluate the candidate lists of destructive moves, constructive moves and swap moves. The swap moves are paired from the 10 best destructive moves and 10 best constructive moves from the current iteration.

The basic evaluator is applied for evaluating all types of moves, while the intermediate evaluator is used for 10 best destructive moves, 10 best constructive moves and 10 best swap moves. Then the advanced evaluator is further employed to correct estimation errors for the 10 best moves of all types at the current iteration. The lowest cost move is selected and executed. If the current solution improves the solution from the previous solution, the search proceeds to the next iteration. Otherwise, the local search heuristic terminates with the current solution.

Since the local search improvement method always stops upon reaching a local optimum, it is very likely to terminate with the same solution for different starting solutions. This accentuates the importance of the method that avoids placing duplicated solutions in the reference set, as described in section 3.5.

3.4 Maintaining And Updating The Reference Set

The Reference Set Update method is an important component in the SS/PR template which keeps records of the b all-time best solutions. Several issues are relevant. First, since the Reference Set is a collection of the top-ranked solutions, it can be implemented as a sorted list. Initially, the list is empty. Then, unique solutions are added into the list and the list is kept sorted on solution evaluations whenever a new solution is added. Once the list is full (i.e., the number of elite solutions in the list reaches its pre-defined limit, of b), the solution currently under consideration is added to the list only if it is better than the current worst solution and does not duplicate any of the other solutions on the list. In this case it replaces the worst solution, and is inserted into the proper position based on its evaluation.

It is critical for the SS/PR heuristic to make sure that the Reference Set does not contains duplicated solutions. The check-for-duplication procedure first checks the values of total cost and total non-ring cost. If two solutions have the same total cost value and the same total non-ring cost value, then their Steiner node vectors are compared against each other to determine if the two solutions are the same. The Reference Set contains only the solutions which are processed by the Improvement Method.

Finally, it is useful to collect some types of statistics throughout the execution of the Reference Set Update method. These statistics include the number of times the Update method is called, as well as the number of times a new solution is added, which we use to control the progress of the SS/PR method. Other auxiliary statistics include a count of the number of partial duplication checks, full duplication checks, and the number of occurrences where duplications were found. These statistics can play important roles in developing further diversification criteria, though they are not implemented in the current application.

3.5 Choosing Subsets of the Reference Solutions

We now describe the method for creating different subsets X of the reference set (denoted as RefSet), as a basis for implementing Step 5 of the SS/PR Template. It is important to note the SS/PR Template prescribes that the set $C(X)$ of combined solutions (i.e., the set of all combined solutions we intend to generate) is produced in its entirety at the point where X is created. Therefore, once a given subset X is created, there is no merit in creating it again. Therefore, we seek a procedure that generates subsets X of $RefSet$ that have useful properties, while avoiding the duplication of subsets previously generated. Our approach for doing this is organized to generate the following four different collections of subsets of RefSet, which we refer to as $SubSetType = 1$, 2, 3 and 4. Let $bNow$ denote the number of solutions currently recorded on $RefSet$, where $bNow$ is not permitted to grow beyond a value $bMax$.

$SubsetType = 1$:	all 2-element subsets.
$SubsetType = 2$:	3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset.
$SubsetType = 3$:	4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset.
$SubsetType = 4$:	the subsets consisting of the best i elements, for $i = 5$ to $bNow$.

We choose the aforementioned four types of subsets of $RefSet$ based on the following reasons. First, 2-element subsets are the foundation of the first "provably optimal" procedures for generating constraint vector combinations in the surrogate constraint setting, whose ideas are the precursors of the ideas that became embodied in scatter search (see, e.g., [4]; [11]). Also, conspicuously, 2-element combinations have for many years dominated the genetic algorithm literature (in "2-parent" combinations for crossover). We extend the 2-element subsets since we anticipate the 3-element subsets will have an influence that likewise is somewhat different than that of the 2-element subsets. However, since the 3-element subsets are much more numerous than the 2-element subsets, we apply an intensification strategy by restricting consideration to those that always contains the best current solution in each such subset. Likewise, we extend the 3-element subsets to 4-element subsets for the same reason, and similarly restrict attention to a sub-collection of these that always includes the two best solutions in each such subset. In addition, to obtain a limited sampling of subsets that contain larger numbers of solutions and achieve an additional intensification effect, we create the special subsets (designated as $SubsetType = 4$), which include the b best solutions as b ranges from 5 to $bMax$.

The methods which create the four types of subsets where $RefSet$ is entirely static (i.e., where $bNow = bMax$ and the set of $bMax$ best solutions never changes) are trivial. However, these algorithms have the deficiency of potentially generating massive numbers of duplications if applied in the dynamic setting (where they must be re-initiated when $RefSet$ becomes modified). Thus we create somewhat more elaborate processes to handle a dynamically changing reference set.

A basic part of the *Subset Generation Method* is the iterative process which supervises the method and calls other subroutines to execute each subset generation method for a given subset type (for $SubsetType = 1$ to

4, then circularly return to 1). Inside each individual subset generation method, once a subset is formed, the solution combination method C(X) (Step 6 of the SS/PR template) is immediately executed to create one or more trial solutions, followed by the execution of the improvement method (Step 7 of the SS/PR template) which undertakes to improve these trial solutions. When these steps find new solutions, not previously generated, that are better than the last (worse) solution in $RefSet$, $RefSet$ must be updated. Since the solution combination method and the improvement method are deterministic, there is no need to generate the same subset X produced at some earlier time. To avoid such duplications, we organize the procedure to make sure that X contains at least one new solution not contained in any subset previously generated.

At the beginning of each iteration, we sort the new solutions in $RefSet$. Any combination of solutions that contains at least one new solution will be generated as a legal subset of $RefSet$ for a given *SubsetType*. The iterative process terminates either when there is no new solution in $RefSet$ ($RefSet$ remains unchanged from the last iteration), or when the cumulative number of executions of the Improvement Method, as it is applied following the solution combination step, exceeds a chosen limit.

3.6 Solution Combination Method

Once a subset of the reference set is determined, we apply a simple solution combination method to produce a series of trial solutions. Let $S*$ denote the subset we consider which contains k distinct vectors (represented by $x(1), \ldots, x(k)$). Then the trial points are produced by the following steps.

(1) For each subset containing $K-1$ vectors, generate the centers of gravity $y(i)$, such that $y(i)_j = \sum_{p \neq i} x(p)_j/(K-1)$ for $i = 1, \ldots, k$ and $j = 1, \ldots, n$.

(2) For each pair $(x(i), y(i))$, consider the general form of the line connecting $x(i)$ and $y(i)$ denoted by $z(w) = x(i) + w(y(i) - x(i))$. We restrict the attention to the two interior points $z(1/3), z(-1/3)$ and two exterior points $z(2/3)$ and $z(4/3)$.

(3) Transform each of the above four points to an 0-1 vector by applying the round-by-threshold rule, that sets the value of an element to 1 if it exceeds a pre-defined threshold u, and set it to 0 otherwise.

Since the trial points are "rounded" by the simple threshold in (3), it is inevitable that different S^* may end up with the same trial vector. These trial vectors are first converted to trial solutions (e.g., by finding a local minimum ring by applying 3-opt on the active Steiner nodes, and calculating the total cost) and then are fed to the *Improvement Method.* Without monitoring, this procedure can generate large numbers of "useless" repetitions by constructing and improving solutions already generated. Therefore, a key issue to produce a highly effective overall heuristic is to avoid such repetitions by subjecting a trial vector to a duplications checking procedure before it is submitted to the constructive and improving heuristics. To do this, we store only the $r = rNow$ most recent solutions generated (allowing $rNow$ to grow to a maximum of $rMax$ different solutions recorded), following a scheme reminiscent of a simple short-term recency memory approach in tabu search. In particular, we keep these solutions in an array $xsave[r]$, $r = 1$ to $rNow$, and also keep track of a pointer $rNext$, which indicates where the next solution will be recorded once the array is full, i.e., once all $rMax$ locations are filled. Let $E0$ and $Hash0$ be the evaluation and hash function value for solution x', and denote associated values for the $xsave[r]$ array by $Esave(r)$ and $Hashsave(r)$. These are accompanied by a "depth" value, which is 0 if no duplication occurs, and otherwise tells how deep in the list - how far back from the last solution recorded - a duplication has been found. For example, depth = 3 indicates that the current solution duplicates a solution that was recorded 3 iterations ago. (This is not entirely accurate, since, for example, depth = 3 could mean the solution was recorded 5 iterations ago and then 2 other duplications occurred, which still results in recording only 3 solutions.) The pseudo code to check for the duplications is shown as follows.

```
Initialization Step:
        $rNow = 0$
        $rNext = 0$
        $CountDup(depth) = 0$, for $depth = 1$ to $rMax$
Duplication Check Subroutine.
Begin Subroutine.
        $depth = 0$
        if $rNow = 0$ then:
             $rNow = 1;\ rNext = 1;$
             $xsave[1] = x'$   (record $x'$ in $xsave[1]$),
             $Esave(1) = E0;\ Firstsave(1) = FirstIndex0$
             Exit the Subroutine
        elseif $rNow > 0$ then:
                (Go through the solutions in "depth order", from the one
most recently stored to the one least recently stored. When a duplication is
found, the loop index r (below) indicates the value of rMax that would have
been large enough to identify the duplication.)
            $i = rNext$
            for $r = 1$ to $rNow$
            if $Esave(i) = E0$ then:
            $qquad$ if $Hash0 = Hashsave(i)$ then:
                    If $x' = x[i]$ then:
                        ($x'$ duplicates a previous solution)
                        depth[i] = r
                        exit the Duplication Check Subroutine
                    Endif
                Endif
            Endif
            $i = i - 1$
            if $i < 1$ then $i = rNow$
        End r
        (Here, no solutions were duplicated by $x'$ . Add $x'$ to the list in
position $rNext$, which will replace the solution previously in $rNext$ if the
list is full.)
        $rNext = rNext + 1$
        If $rNext > rMax$ then $rNext = 1$
        If $rNow < rMax$ then $rNow = rNow + 1$
        $xsave[rNext] = x'$
        $Esave(rNext) = E0$
        $Hashsave(rNext) = Hash0$
    Endif
    End of Duplication Check Subroutine
```

4 Computational Results

To provide a direct comparison with results of Xu, Chiu and Glover [19], we report our computational outcomes for a sets of test problems generated randomly from distributions whose parameters are selected to create the most difficult problem instances. Two set of such problems were examined in [19]. The first set of test problems is restricted to problems of relatively small dimensions so they were capable of being solved by the exact branch and cut approach method. Problems from the second test set have larger dimensions, and are beyond the ability of current exact methods to solve. In this paper, we focus on the second test set which contains harder problem instances than the first test set. We report the results produced by our SS/PR method and compare them to those obtained with our TS method. In the following tables, we represent the problem dimensions by m and n, which identify the number of target and Steiner nodes respectively.

4.1 Parameter Description

In addition to the parameter setting of the tabu search heuristic (which we describe in Section 2), we have few parameters to choose for the scatter search heuristic. The maximum number of solution in Reference Set is set to be 30, the h value is set to 5, and threshold value for rounding the trial points is set to 0.75. The heuristic terminates either after 100 iterations, or there is no change for reference set after one iteration. Since our problem sizes are big, it makes the regular diversification generator create many inferior solutions which possess too many active Steiner nodes, therefore affects the overall performance of the algorithm. To overcome this, we replace the regular diversification generator by the random start method, that is, instead of using the regular diversification generator to produce the initial pool of reference set, we fill the initial reference set with the solutions whose active Steiner nodes are selected randomly and only 10% of the total Steiner nodes are selected as active. However, we need to point out that the regular diversification generator works more effectively than the random start method for the STS problem, as reported in our previous research.

All parameters values are selected intuitively or based on several preliminary experiments, without any attempt at fine tuning. An effort to fine-tune these parameters can be based, for example, on a systematic statistical testing procedure (see [18]), and may significantly improve the performance of our algorithm. Compared with the tabu search algorithm, the SS/PR heuristic incorporates much less numbers parameters to be selected and fine-tuned.

4.2 Test Results

We test our scatter search heuristic and the tabu search heuristic, and list the results on the second set in Table 1. The results obtained by the scatter search heuristic are listed in the "SS" column while the tabu search results are listed in the "TS" column.

Problem	TS		SS	
$(m \times n)$	Cost	CPU (min.)	Cost	CPU (min.)
100×100	16740	0:48	16740	2:35
150×100	20109	1:52	20172	8:49
200×100	25703	2:49	25703	12:01
125×125	16811	1:54	16811	9:09
175×125	21693	2:30	21693	13:34
225×125	26735	2:44	26735	12:17
150×150	19874	2:03	19874	11:59
200×150	24944	3:08	26928	30:01
250×150	29001	4:12	31138	38:21
175×175	21657	3:27	21657	24:41
225×175	25653	3:56	25653	29:12
275×175	28267	5:00	28267	19:03
200×200	23418	3:32	23418	32:38
250×200	26920	5:48	26920	35:19
300×200	30518	5:55	30518	41:21
250×250	26170	6:44	26170	43:23
300×250	29821	9:09	29821	50:45
350×250	32772	11:41	32772	52:47
100×300	13584	2:14	13584	11:45
200×300	21825	6:02	21825	16:55
300×300	29193	10:33	29193	59:32

Table 1: Computational Results on Larger Size Problems

From Table 1, we observe that the indicated implementation of the SS/PR method can yield nearly the same solution quality as the TS method. It ties 18 solutions out of the 21 test problems with the TS method, and produces solutions marginally inferior to those given by the TS method for the remaining three cases. However, our SS heuristic takes somewhat more CPU execution times than the TS method. This efficiency gap comes from the fact that the embedded TSP problem in SRS is time-consuming to evaluate (our statistics show that 97% of the execution time is spent on the local improvement method), and the recency-based TS memory is more effective to avoid unnecessary move evaluation.

One of the important findings in our prior research on the STS (see [20]) is that the performance of SS/PR can potentially be improved using a customized solution combination method. More specifically, we replace the threshold rounding rule by designating the ith component of the trial point to receive the assignment $x_i = 1$ if and only if at least t of its r parents have

$x_i = 1$. For example, we test the following rules which simplify the creation of the trial point x from the 2-element subset (with two parents y and z):

(1) $x_i = 1$ if $y_i = 1$ and $z_i = 1$;

(2) $x_i = 1$ if $y_i = 1$ and $z_i = 0$;

(3) $x_i = 1$ if $z_i = 1$ and $y_i = 0$.

We report the results from the above three tests (marked as SS1, SS2 and SS3) in Table 2 and we also provide a comparison with our SS results in the same table.

Problem	SS		SS1		SS2		SS3	
$(m \times n)$	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
100×100	16740	2:35	16740	2:04	16740	1:32	16740	1:35
150×100	20172	8:49	20172	5:28	20172	4:23	20172	4:12
200×100	25703	12:01	25703	10:01	25742	8:33	25812	8:13
125×125	16811	9:09	16811	7:49	16811	7:01	16811	6:52
175×125	21693	13:40	21693	10:44	21693	8:56	21693	9:11
225×125	26735	12:17	26735	8:57	27003	8:02	27003	7:54
150×150	19874	11:59	19874	9:22	19874	8:21	19923	8:03
200×150	26928	30:01	26928	24:37	26928	21:20	26928	21:45
250×150	31138	38:21	31138	31:29	31138	29:48	31138	29:56
175×175	21657	24:41	21657	20:14	22343	18:32	22276	18:55
225×175	25653	29:12	25653	23:41	25653	21:37	25653	20:32
275×175	28267	19:03	28267	16:33	28267	13:02	28267	14:38
200×200	23418	32:38	23418	29:38	23418	25:25	23418	25:12
250×200	26920	35:19	26920	31:01	26920	28:47	26920	28:27
300×200	30518	41:21	30518	36:35	30518	31:01	30518	30:47
250×250	26170	43:23	26170	36:13	26170	33:29	26170	34:45
300×250	29821	50:45	29821	40:45	29886	35:33	29886	33:58
350×250	32772	52:47	32772	40:47	32772	38:01	32772	38:44
100×300	13584	11:45	13584	9:55	13584	8:02	13584	7:59
200×300	21825	16:55	21825	12:13	21825	10:33	21825	10:21
300×300	29193	59:32	29193	47:51	30004	43:26	29839	44:55

Table 2: Comparisons of results with the simplified solution combination rules (the time unit of CPU is min.)

Table 2 produces results very similar to those shown in [20], thus providing a useful improvement over the outcome shown in Table 1. All three simplified rules can effectively reduce the execution time of the SS method. SS1 obtains the same solution quality as SS does. SS2 and SS3 can produce greater savings in time at the expense of five inferior solutions. Our analysis in [20] shows that the rules (2) and (3) tend to produce more assignments of $x_i = 1$, therefore causing the method to evaluate and execute more destructive moves which are less expensive.

5 Conclusion

In this paper, we have described a variant of the Steiner tree family, the "Steiner Ring-Star" problem with application to leased-line network design. The problem involves selecting a subset of hubs to form a least-cost ring backbone network, while connecting each customer site to one of the selected hubs. We have reviewed an advanced tabu search algorithm which is documented as the best heuristic available for the SRS problem.

Our purpose has been to develop and test a scatter search method for the SRS to determine the potential of this procedure for becoming a strategic component of a more advanced method that melds SS with TS. This first level testing to discover the independent strength of the SS approach also has the utility of expanding our knowledge about the type of performance that is likely to result in applying SS on its own in other settings. In addition, it affords an opportunity to verify whether previous findings about SS in the context of STS problems are supported in the more difficult and challenging environment of the SRS problem.

The outcomes were extremely encouraging. Not only does the SS method perform well, but the outcomes from its independent operation come remarkably close to matching those of the TS method, both in solution quality and execution time. The same type of strategy that improved execution time in the STS context also proved its merit in the current SRS setting, although with some degradation of solution quality in the case of two variants other than the best variant we devised.

In sum, we conclude that the SS approach is a potentially strong partner for TS for the purpose of creating a more advanced method, and is also a highly viable solution procedure in its own right. By endowing the SS framework with the ability to take advantage of memory-based processes such as provided by TS, the prospects appear promising for achieving additional gains. Such issues provide an inviting area for future research.

References

[1] S. Chopra and M. R. Rao, On the Steiner Tree Problem I & II, Mathematical Programming, 64 (1994), pp. 209-246.

[2] C. Duin, and S. Voβ, Steiner Tree Heuristics - A Survey, i n: *Operations Research Proceedings 1993, Papers of the 22nd Annual Meeting of DG OR in Cooperation with NSOR*, pp. 485-496, (Springer-Verlag, 1994).

[3] F. Glover, Parametric Combinations of Local Job Shop Rules, *Chapter IV, ONR Research Memorandum no. 117, GSIA*, (Carnegie Mellon University, Pittsburgh, PA ,963).

[4] F. Glover, A Multiphase Dual Algorithm for the Zero-One Integer Programming Problem, *Operations Research*, 13, 6 (1965), pp. 879.

[5] F. Glover, Heuristics for Integer Programming Using Surrogate Constraints, *Decision Sciences*, 8 (1977), pp. 156-166.

[6] F. Glover, Tabu Search - Part I, *ORSA Journal of Computing*, 3 (1989), pp. 190-206.

[7] F. Glover, Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges, in: *Interfaces in Computer Science and Operations Research*, Barr, Helgason and Kennington, eds., (Kluwer Academic Publishers, 1996), pp. 1-75.

[8] F. Glover, Scatter Search and Path Relinking, in: *New Methods in Optimization*, D. Corne, M. Dorigo and F. Glover (Eds.), (McGraw-Hill, 1999).

[9] F. Glover and M. Laguna *Tabu Search*, (Kluwer Academic Publishers, 1997)

[10] F. Glover and C. McMillan, The General Employee Scheduling Problem: An Integration of Management Science and Artificial Intelligence, *Computers and Operations Research*, Vol. 15, No. 5 (1986), pp. 563-593.

[11] H. J. Greenberg and W.P. Pierskalla, Surrogate Mathematical Programs, *Operations Research*, 18 (1970), pp. 924-939.

[12] Y. Lee, L. Lu, Y. Qiu and F. Glover, Strong Formulations and Cutting Planes for Designing Digital Data Service Networks, *Telecommunication Systems*, **2** (1994), pp. 261-274.

[13] Y. Lee, S.Y. Chiu and J. Ryan, A Branch and Cut Algorithm for a Steiner Tree-Star Problem, *INFORMS Journal on Computing*, Vol. 8, No. 3 (1996), pp. 194-201.

[14] Y. Lee, S.Y. Chiu and J. Ryan, A Branch and Cut Algorithm for a Steiner Ring-Star Problem, *Working Paper*, (U S WEST Advanced Technologies Inc., Boulder, CO., 1996).

[15] J. Xu, S. Y. Chiu and F. Glover, Using Tabu Search to Solve the Steiner Tree-Star Problem in Telecommunications Network Design, *Telecommunication Systems*, 6 (1996), pp. 117-125.

[16] J. Xu, S. Y. Chiu and F. Glover, Probabilistic Tabu Search for Telecommunications Network Design, *Combinatorial Optimization: Theory and Practice*, Vol. 1, No. 1 (1996), pp. 69-94.

[17] J. Xu, S. Y. Chiu and F. Glover, Tabu Search for Dynamic Routing Communications Network Design, *Telecommunication Systems*, 8 (1997), pp. 55-77.

[18] J. Xu, S. Y. Chiu and F. Glover, Fine-Tuning a Tabu Search Algorithm with Statistical Tests, *International Transactions in Operational Research*, 5 (1998), pp. 233-244.

[19] J. Xu, S. Y. Chiu and F. Glover, Optimizing a Ring-Based Private Line Telecommunication Network Using Tabu Search, *Management Science*, Vol. 45, No. 3 (1998), pp. 330-345.

[20] J. Xu, S. Y. Chiu and F. Glover, Tabu Search and Evolutionary Scatter Search for "Tree-Star" Network Problems, with Applications to Leased-LIne Network Design, to appear in *Telecommunications Optimization*, David Corne(Ed.) (Wiley, 2000).

[21] J. Xu, and J. P. Kelly, A New Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem, *Transportation Sciences*, vol. 30,No. 4 (1996), pp. 379-393.

Appendix A Mathematical Formulation for the SRS Problem

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

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

subject to:

$$\sum_{j\in N} x_{ij} = 1, \qquad i \in M, \tag{2}$$

$$x_{ij} \leq z_j, \qquad i \in M, \quad j \in N, \tag{3}$$

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

$$\sum_{k\in N} y_{jk} + \sum_{k\in N} y_{kj} = 2z_j, \qquad j \in N, \tag{5}$$

$$\sum_{j \in H} \sum_{k \in H} y_{jk} \leq \sum_{j \in \{H-l\}} z_j + 1 - z_t, \quad l \in H, \quad H \subset N, \quad |H| \geq 3, \quad t \in N - H, \tag{6}$$

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

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

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

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.