FINE-TUNING A TABU SEARCH ALGORITHM WITH STATISTICAL TESTS

Jiefeng Xu[a,c], Steve Y. Chiu[b] and Fred Glover[c]

[a] Delta Technologies, Inc., 1001 International Boulevard, Atlanta, GA 30354, U.S.A.

[b] GTE Laboratories, Inc., 40 Sylvan Road, Waltham, MA 02254, U.S.A.

[c] Graduate School of Business, University of Colorado at Boulder, CO 80309-0419, U.S.A.

Abstract. Tabu Search is a metaheuristic that has proven to be very effective for solving various types of combinatorial optimization problems. To achieve the best results with a tabu search algorithm, significant benefits can sometimes be gained by determining preferred values for certain search parameters such as tabu tenures, move selection probabilities, the timing and structure of elite solution recovery for intensification, etc. In this paper, we present and implement some new ideas for fine-tuning a tabu search algorithm using statistical tests. Although the focus of this work is to improve a particular tabu search algorithm developed for solving a telecommunications network design problem, the implications are quite general. The same ideas and procedures can easily be adapted and applied to other tabu search algorithms as well.

Key words. Tabu Search Heuristic , Statistical Test, Telecommunications Network Design.

This research was supported in part by the Air Force Office of Scientific Research AASERT grant #F49620-92-J-0248.DEF.

Published in *International Transactions in Operational Research*, **5** (1998), pp. 233-244.

1. Introduction

Tabu Search (TS) is a metaheuristic method that has proven to be very effective for many combinatorial optimization problems. It employs *adaptive memory* and *responsive exploration* to effectively search the solution space and to prevent it from being trapped in a local optimum. (see [3] and [4] for comprehensive overviews of the TS method). Typically, a TS algorithm involves a set of parameters or options that need to be set appropriately in order to achieve the best results. Such parameters or options include tabu tenures, move selection probabilities, recovery strategies for intensification, etc. The process of fine-tuning these parameters is often time-consuming and involves extensive computational experiments.

Barr et al. in [1] address the important issues of designing and reporting on computational experiments with heuristic methods, and consider that "the selection of parameter values that drive heuristics is itself a scientific endeavor, and deserves more attention than it has received in the operations research literature". The authors further claim that fining-tuning a heuristic is an area where the scientific method and statistical analysis could and should be employed. This paper presents a first attempt to systematically fine-tune a TS algorithm using statistical analysis. We employ some standard statistical tests

and experimentation design techniques to improve a specific TS algorithm that was proposed by Xu, Chiu and Glover in [9]. Although the focus of this paper is to improve the performance of this particular algorithm, the implications are general. The same ideas and procedures can easily be employed to improve other heuristic methods as well.

The problem that the TS algorithm attempts to solve was motivated by a real-world telecommunications network design problem which was described in [7]. The problem can be briefly described as follows. Consider a simple graph $G = (V, E)$, where the set of nodes V is partitioned into two disjoint subsets S and T, and the set of edges $E = S \times T \cup S \times S$. The nodes in S are called *Steiner nodes* and the nodes in T are called *target nodes*. In a feasible network design, each Steiner node is either selected or unselected and each target node has to be connected to exactly one selected Steiner node directly by an edge. The selected Steiner nodes (called *active* nodes) must also be interconnected to form a spanning tree themselves. Now suppose that there is a cost associated with each edge in E and each node in S. We want to determine the optimal network design that minimizes the total edge and node costs. The problem is referred to as the Steiner Tree-Star problem (STS). A precise mathematical formulation of the problem is given in Appendix A.

This paper is organized as follows. Section 2 describes a probabilistic TS algorithm that was first proposed in [9] for solving the network design problem. This is the algorithm that we want to improve by using the statistical tests in

this paper. Section 3 introduces two specific statistical tests that will be used to fine-tune the TS algorithm. A tree based search procedure for the best tabu parameter values is also described. Section 4 addresses the implementation issues and reports the computational results. Various options for improving the algorithm are examined in detail in this section. Finally, Section 5 summaries our findings and concludes the paper.

2. The Probabilistic TS Based Heuristic

In this section, we briefly outline the algorithm that was proposed in [9]. The interested readers are referred to that paper for details. Our TS algorithm starts at an arbitrary initial solution. At each iteration, a set of candidate neighborhood moves is evaluated and a "best" move is selected. A new solution is then generated by taking the selected move. During each iteration, certain neighborhood moves will be considered as *tabu* moves and are thus excluded from the candidate list. Typically, a non-tabu move with "good" evaluation (which can be determined either deterministically or probabilistically) is selected, although aspiration criteria can allow us to select a tabu move if it is particularly attractive. The algorithm terminates when a pre-defined number of iterations elapses. We now provide a brief discussion of the following five major components of the algorithm: neighborhood structure and moves, move

evaluation and error correction, TS memories, probabilistic move selection, and advanced restarting and recovery.

Neighborhood structure. We consider three types of moves. Those of the first type are called constructive moves since they add a currently inactive Steiner node to the current solution. Once the set of active Steiner nodes is determined, a feasible solution can easily be constructed by connecting the active Steiner nodes using a spanning tree and by linking the target nodes to their nearest active Steiner nodes. Moves of the second type are called destructive moves since they remove a currently active Steiner node from the current solution. The third type of moves, called swap moves, exchange an active Steiner node with an inactive Steiner node. Clearly, the swap moves induce a more significant change in the current solution and hence require a more complex evaluation. For this reason, we execute the swap moves less frequently — once for every n_1 iterations (for perturbation) and consecutively n_2 times when the search fails to improve the current solution for a pre-specified number of iterations (for intensification).

Move evaluation and error correction. To evaluate a potential move, we need to estimate the cost of the resulting new solution. For a constructive move, we simply connect the new Steiner node to the closest active Steiner node in the current solution to form a new solution. For a destructive move, we consider only those active Steiner nodes with degree less than or equal to three in the current solution since the removal of such a node creates at most three

disconnected components, and a new solution can easily be reconstructed. The swap move can be viewed as a combination of the constructive and destructive moves. The error introduced by the preceding estimates can be corrected by running a minimum spanning tree algorithm. We apply this error correction procedure every few iterations and also whenever a new "best" solution is found. Throughout the algorithm, we maintain a set of elite solutions that represent the "best" solutions found so far. The error correction procedure is also applied to these solutions periodically.

TS memory. Our TS algorithm uses both a short term memory and a long term memory to prevent the search from being trapped in a local minimum and to intensify and diversify the search. The short term memory operates by imposing restrictions on the set of solution attributes that are permitted to be incorporated in (or "changed by") candidate moves. More precisely, a node added to the solution by a constructive move is prevented from being deleted for a certain number of iterations, and likewise a node dropped from the solution by a destructive move is prevented from being added for a certain (different) number of iterations. For constructive and destructive moves, therefore, these restrictions ensure that the changes caused by each move will not be "reversed" for the next few iterations. For each swap move, we impose tabu restrictions that affect both added and dropped nodes.

The number of iterations during which a node remains subject to a tabu restriction is called the *tabu tenure* of the node. We establish a relatively small

range for the tabu tenure, which depends on the type of move considered, and each time a move is executed, we select a specific tenure randomly from the associated range. A move is classified tabu if it is prevented as a result of the tabu status of its attributes (the nodes added or dropped, according to the case). In more complex settings, the tabu classification can be a function of the tabu status of multiple attributes. We also use an aspiration criterion to override the tabu classification whenever the move will lead to a "very good" solution.

The long term memory is a frequency based memory that depends on the number of times each particular node has been added or dropped from the solution. We use this type of memory to discourage type of changes that have already occurred frequently (and consequently to encourage changes that have occurred less frequently). This represents a particular form of frequency memory based on attribute transitions (changes). Another type of frequency memory is based on residence, i.e., the number of iterations that nodes remain in or out of solution.

Probabilistic choice. At each iteration, a "best" candidate move is selected probabilistically as suggested in [5]. First, all neighborhood moves (including tabu moves) are evaluated. If the move with the highest evaluation satisfies the aspiration criterion, it will be selected. Otherwise, we consider the list of moves ordered by their evaluations. For this purpose, tabu moves are considered to be moves with highly penalized evaluations. We select the top move with a

probability p and reject the move with probability $1-p$. If it turned out that we rejected a move, then we consider the next move on the list in the same fashion. If it turned out that no move was selected at the end of this process, we will select the top move anyway. In practice, the number of moves that need to be considered can be reduced to a relatively small number d because the probability of selecting one of the d best moves is very high (equal to $1-(1-p)^d$). We can also make the selection probability vary with the quality of the move by changing it to $p^{\beta_1 r-\beta_2}$, where r is the ratio of the current move evaluation to the value of the best solution found so far, and β_1 and β_2 are two positive parameters. The new fine-tuned probability will increase the chance of selecting "good" moves.

Intensification by recovery. We implement a variant of the advanced restarting and recovery strategy in which the recovery of the elite solution is postponed until the last stage of the search. The elite solutions, the K best solutions found so far, are recovered in reverse order from the worst solution to the best solution. The recovery of each solution launches a search that constitutes a fixed number of iterations. The worst elite solution is replaced immediately if a better solution is found. After each solution is recovered, all tabu restrictions are removed and reinitialized. (Some TS recovery strategies, which we do not consider here, also recover portions of the memory associated with the solutions recovered.)

3. Fine-Tuning TS Algorithm with Statistical Tests

In this section, we describe a procedure for fine-tuning a TS algorithm based on statistical tests. For a given TS algorithm, we define *factors* to be the independent parameters or options whose values need to be determined, and define *treatments* to be the candidate settings (values) for any given factor. All specific combinations of factors and treatments are always tested on the same set of representative problem instances called test problems. We examine the factors sequentially according to their *a priori* importance and attempt to find the "best" treatment for each factor based on the test results. (Note that the order in which factors are examined is important since it may impact the total number of tests required for our fine-tuning procedure.) However, direct comparisons between different runs are always difficult if the results from one run do not completely dominate those from the other runs for all test problems. Since the underlying distribution of the test results is often unknown, comparisons of the average test results is also imperfect. Therefore we introduce two statistical tests to analyze the test results under a certain confidence level.

Friedman's Test. If we assume for a given factor, the "noise" in test results yielded by various treatments is drawn independently from the same continuous unknown distribution, then Friedman's test can be used to test the significance of the "treatment effect". The null hypothesis is no treatment effect, that

is, the difference in test results is caused by the randomness rather than by different treatments. The readers are referred to [2] and [6] for details about Friedman's test. Let x_{ij} be the test result yielded by treatment i for test problem j for $i = 1, \ldots, I$ and $j = 1, \ldots, J$. For each problem j, rank the results x_{ij} $(i = 1, \ldots, I)$ from 1 to I according to their goodness. Let R_{ij} denote the rank of x_{ij}. (If ties occur, each tied entry receives the same rank, equal to the average of the ranks received by the tied entries as if the ties were broken arbitrarily. For example, the rank vector for the test results (100,99,100,101) is (2.5,1,2.5,4)). Then the test statistic is computed by:

$$F_r = 12\sum_{i=1}^{I} R_{i.}^2/(IJ(I+1)) - 3J(I+1)$$

where $R_{i.} = \sum_{j=1}^{J} R_{ij}$. In practice, for moderate values of J, the test statistic F_r has approximately a chi-squared distribution with $I-1$ degrees of freedom when the null hypothesis is true. Therefore, we reject the null hypothesis if the computed test statistic exceeds the critical value $\chi^2_{\alpha,I-1}$ at the confidence level $1-\alpha$.

If the null hypothesis is rejected in Friedman's test, which means the various parameter settings will have an effect on the performance of the algorithm, a comparison between any two treatments i_1 and i_2 is conducted to see if one dominates the other. The treatment i_1 is considered to be better (with smaller mean and rank sum) than treatment i_2 at the confident level $1-\alpha$ if

$$R_{i_2.} \geq R_{i_1.} + z_\alpha * \sqrt{JI(I+1)/6}$$

where z_α is the $100(1-\alpha)th$ percentile of the standard normal distribution.

To find the best treatment for a factor, we only need to compare each treatment to the one with the smallest rank sum $R_{i_{min}.}$. Thus, if $R_{i.} \geq R_{i_{min}.} + z_\alpha * \sqrt{JI(I+1)/6}$, we consider treatment i to be inferior to the treatment i_{min} and eliminate it from future experiments. Otherwise, we accept treatment i as one of the possible best settings for this factor. $I-1$ comparisons are required to find the best treatments.

Wilcoxon's Test for Paired Observations. In this test, we compare two independent runs that have different treatments for the factors previously tested. Since both runs are tested on the same problem set, we assume that both test result series have continuous distributions that differ only with respect to their means. This assumption will allow us to use the Wilcoxon signed-ranked test (see [2]) to see if the difference between the two runs is significant. Assume that runs A and B yield the result series X_A and X_B respectively. Then the null hypothesis is $H_0 : \mu_D = 0$, where $D = X_A - X_B$ and μ_D represents the mean of D.

The test employs the signed-ranked statistic s_+, which can be calculated as follows: first disregard the signs of the components of D corresponding to the test problems, and rank the components in order of increasing magnitude of their absolute values. Then calculate s_+ as the sum of the ranks associated with the positive D components (where $s_+ = 0$ if all components are nonpositive).

We reject H_0 at the confidence level $1-\alpha$ when $s_+ \geq d_1$, or when $s_+ \leq J(J+1)/2 - d_1$, where d_1 is the upper-tail critical value which can be obtained as described in [2]. If H_0 is rejected by the test and $s_+ \geq d_1$, we conclude that the mean of X_A is greater than that of X_B, which implies that run B outperforms run A. Likewise, if $s_+ \leq J(J+1)/2 - d_1$, then we conclude that run A dominates run B.

Now suppose there are several factors for tests and each factor involves several treatments. We attempt to find the overall best factor-treatment configurations for the algorithm. Assuming the covariance between various factors is limited, we propose the following procedure for searching the best configurations. If the covariance is significantly large, then more complicated experimental design techniques such as Latin Square Design (see [8]) may be required. More precisely, we assume that the factors exhibit a conditional dominance property where, under the ordering in which they are considered (arranged by decreasing importance), if a set of test outcomes dominates another at a given level, each descendant of the first set of outcomes will also dominate each descendant of the other set of outcomes at higher levels. Small covariances provide an indication that such an assumption is likely to hold.

Tree Growing and Pruning Method.

Step 1 Suppose that there are k factors for tests and they are ranked in order of decreasing importance, $F = (F_1, F_2, \ldots, F_k)$. Chose a set of treatments, $T_i = (T_{i1}, T_{i2}, \ldots, T_{it_i})$, for each factor F_i.

Step 2 Initiate the search tree by considering factor F_1 at the root node, which is also considered the unique initial leaf node. Define level $i = 1$.

Step 3 For every currently existing leaf node at level i, grow t_i branches where each branch represents one of the treatments in T_i. Run the test problems for each branch and collect the test results, thereby creating a leaf node at level $i + 1$. The nodes at level i lose their status as leaf nodes.

Step 4 For each set of leaf nodes (at level $i + 1$) that share the same parent node, use Friedman's test to determine the best treatments. Prune (eliminate) any leaf nodes that are inferior to the best treatments.

Step 5 For each pair of currently remaining leaf nodes that do not share a parent node, and which has not yet been examined at this step, apply the Wilcoxon test. If the members of the pair under investigation are significantly different, prune the one which is inferior.

Step 6 $i = i + 1$. If $i > k$, go to *Step 7*; else go to *Step 3*.

Step 7 Every currently remaining leaf node represents one of the best configurations for the algorithm. Terminate.

At each level I of the foregoing procedure, the ith factor is considered, and Friedman's test and Wilcoxon's test are used to eliminate the inferior treatments from future consideration. We illustrate this procedure with our application in the next section. It should be noted that even if the assumptions underlying the procedure are not entirely validated, so that the screening tests are not guaranteed to yield the dominating sets of treatments at the chosen confidence levels, these tests nevertheless afford a strategy to yield "good" sets of treatments.

4. Computational Results

The TS algorithm with various factor-treatment configurations were tested on nineteen test problems generated in [9]. These are STS problems defined on grid graphs which were shown to be very hard to solve. The TS algorithm with the specific factor-treatment configuration described in [9] serves as a base algorithm for applying the fine-tuning design of this study. We measure the benefit derived from our fine-tuning strategies in relation to the results obtained by the base algorithm (though of course worse algorithmic instances might presumably be chosen for comparison, which would increase the relative attractiveness of our improvements). For each run of the algorithm over the nineteen problems, we report the results as the percentage improvement of the objective function values produced by the current run over those produced by the base algorithm. This normalization gives a conservative basis for comparison that eliminates the differences in solution magnitudes between different test problems.

In this study, we fine-tune the following five factors ordered by their importance (conceived in a conditional sense rather than an absolute sense): the tabu tenures, the base probability for move selection, the fine-tuned probability for move selection, the recovery strategy, and the frequency of activating swap moves.

The base algorithm is designated as *Run* 0 and each subsequent run is designated as *Run* 1, *Run* 2, ..., etc. For the sake of brevity, only the relevant rank sums or test statistics are reported. We do not report the computational times of the different runs since they are insensitive to the five factors.

Tabu Tenure. In the base algorithm, the tabu tenures are generated randomly within certain intervals that depend on the move types. Since a constructive move of adding a node to the active node set (denoted as A) introduces a fixed cost, and thus makes the move appear less attractive than a destructive move, a longer tabu tenure is assigned to avoid destructive moves. In the base algorithm, the tabu tenure interval is [1,3] for the constructive moves, [2,5] for the destructive moves. A tenure of [1,3] is assigned to the two constituent constructive and destructive moves that together compose swap moves. These choices appeared to be quite effective in our previous study [9].

In this study we examine a different criterion that assigns tabu tenures based on the relative sizes of A and $\bar{\mathsf{A}}(= \mathsf{S} - \mathsf{A})$. That is, longer tenures are given to moving a node from the larger set to the smaller set, because there are more options for these moves than for moves of the opposite type. Let $random(a, b)$ denote the function that returns an integer between a and b randomly, and let $|S|$ be the cardinality of the set S. Then the tabu tenure in this study is generated as follows:

$\mathsf{tabu\_drop\_tenure} = \max\{1, \mathsf{random}(\mathsf{c}_1 * |\mathsf{A}|/|\bar{\mathsf{A}}|, \bar{\mathsf{c}_1} * |\mathsf{A}|/|\bar{\mathsf{A}}|)\}$ (for the constructive move of adding node x, to prevent x from being dropped),

$\mathsf{tabu\_add\_tenure} = \max\{1, \mathsf{random}(\mathsf{c}_2 * |\bar{\mathsf{A}}|/|\mathsf{A}|, \bar{\mathsf{c}_2} * |\bar{\mathsf{A}}|/|\mathsf{A}|)\}$ (for the destructive move of dropping node y, to prevent y from being added).

As in the base algorithm, the tabu tenures for the two constituent moves that compose the swap moves are set to be the same as that for *tabu_drop_tenure*. Because the ratio of $|A|/|\bar{A}|$ and $\bar{A}|/|A|$ may be fractional, we need to examine a range of values of $c_1, \bar{c_1}, c_2$ and $\bar{c_2}$ to determine the best setting. Thus, eighteen candidate settings (*Run* $1, \ldots,$ *Run* 18) are tested. Their values and the resulting rank sums are presented in Table 1.

Run	1	2	3	4	5	6	7	8	9
$(c_1, \bar{c_1})$	(2,5)	(1,3)	(1,3)	(2,5)	(2,5)	(3,7)	(3,7)	(5,9)	(3,7)
$(c_2, \bar{c_2})$	(1,3)	(1,3)	(2,5)	(2,5)	(3,7)	(2,5)	(3,7)	(3,7)	(5,9)
$R_{i.}$	271.5	229.5	228.5	234.5	261	201	185.5	206.5	191
Run	10	11	12	13	14	15	16	17	18
$(c_1, \bar{c_1})$	(5,9)	(7,11)	(7,11)	(5,9)	(9,13)	(9,13)	(7,11)	(11,15)	(11,15)
$(c_2, \bar{c_2})$	(5,9)	(5,9)	(7,11)	(7,11)	(7,11)	(9,13)	(9,13)	(9,13)	(11,15)
$R_{i.}$	160.5	154.5	143	155.5	154.5	89	132	142.5	108.5

Table 1: Tests on Tabu Tenures

From Table 1, we compute Friedman's test statistic $f_r = 82.640$ which is greater than $\chi^2_{0.1,17} = 24.769$ and discloses that the above eighteen settings will have an effect on the outcome at the confidence level 0.90. Since *Run* 15 has the minimum rank sum, we prune each other *Run* i at the confidence level 0.90 such that $R_{i.} \geq R_{i_{min}.} + z_\alpha * \sqrt{JI(I+1)/6} = 131.12$. Therefore, only *Run* 15 and *Run* 18 survive for future consideration.

From these two best settings, we find that there is no need to discriminate among the constant terms ($(c_1, \bar{c_1}$ versus $(c_2, \bar{c_2})$) for different move types

because the ratio terms make much more significant differences and thereby impose greater restrictions for moves that transferred a node from a larger set to a smaller set. Furthermore, the lower and upper bounds defining the intervals from which tabu tenures are drawn preferably are larger than those previously suggested in [4]. Part of reason for this comes from our making use of the relative size ratio in the current design. The larger bounds produce more appropriate tabu tenures when the ratio is a small fractional number.

Base Probability for Move Selection. Here we attempt to determine whether the probabilistic TS choice rule outperforms the deterministic TS choice rule, (which simply chooses the best non-tabu move if the aspiration criterion is not satisfied), and, if so, to determine the best value for the base probability p in the probabilistic TS algorithm.

In order to see the effects produced by the different values of the base probability p, we disable the fine-tuned probability function in the base algorithm (this implies $\beta_1 = 0$ and $\beta_2 = -1.0$) and choose the following six treatments for the factor p: 0.25, 0.28, 0.30, 0.32, 0.35 and 1.0. The first five treatments are possible p values clustered around 0.3 (the most commonly used p value), and the last treatment actually represents the deterministic TS algorithm. For each of the unpruned nodes *Run* 15 and *Run* 18, we initiate six branches. The rank sums are summarized in Tables 2 and 3 respectively.

Run	19	20	21	22	23	24
p	0.25	0.28	0.30	0.32	0.35	1.0
$R_{i.}$	58.5	42.5	72.5	58.0	69.0	98.5

Table 2: Tests on p for *Run* 15

Run	25	26	27	28	29	30
p	0.25	0.28	0.30	0.32	0.35	1.0
$R_{i.}$	57.0	67.0	59	58.5	63.5	94.0

Table 3: Tests on p for *Run* 18

From Table 2, Friedman's test statistic is $f_r = 26.740 > \chi^2_{0.1,6} = 9.236$, which means that the value of p will affect the outcome at the confidence level 0.90. Since *Run* 20 has the minimum rank sum, we prune each *Run* i at the confidence level 0.90 such that $R_{i.} \geq R_{i_{min}.} + z_\alpha * \sqrt{JI(I+1)/6} = 56.91$. Therefore, all runs except *Run* 20 are pruned. In addition, we observe that the rank sum from *Run* 24 is significantly larger than that of *Run* 20, implying that the probabilistic TS approach strongly dominates the deterministic TS approach in this setting.

Friedman's test statistic from Table 3 is $f_r = 14.677$, also confirming the evident influence exerted by the p values. However, this time only *Run* 30 is pruned. For *Runs* 25 through 29, *Run* 25 has the minimum rank sum, but it is not small enough to dominate the others.

We now have six remaining nodes at this level, one descending from *Run* 15 and the other five from *Run* 18. We apply the Wilcoxon test to the resulting five pairs. The test statistics are listed in Table 4.

Pair	(20, 25)	(20, 26)	(20, 27)	(20, 28)	(20,29)
s_+	128	149	136	130	132

Table 4: Wilcoxon Tests on p

The critical value d_1 for $\alpha = 0.098$ is 128 (see [2]). Therefore, all runs except *Run* 20 are pruned for $\alpha = 0.1$ and p is now fixed to 0.28.

Fine-tuned Probability for Move Selection. We devise seven treatments to search for the best settings of β_1 and β_2. The last six treatments represent various combinations of β_1 and β_2 values while the first treatment ($\beta_1 = 0$ and $\beta_2 = -1.0$) represents the case without a fine-tuned option (*Run* 20). Table 5 summarizes the rank sums obtained.

Run	20	31	32	33	34	35	36
β_1	0	1.0	1.1	0.9	1.0	1.1	0.9
β_2	-1.0	0.15	0.15	0.15	0.20	0.20	0.20
$R_{i.}$	67.0	71.5	95.0	49.0	93.0	77.5	79.0

Table 5: Tests on β_1 and β_2

From Table 5, we compute Friedman's test statistic $f_r = 16.821 > \chi^2_{0.1,6} = 10.645$. This treatment effect is significant at the confidence level 0.90. Since *Run* 33 has the minimum rank sum, we prune any runs at the confidence level 0.90 if their rank sums are no less than $R_{i_{min.}} + z_\alpha * \sqrt{JI(I+1)/6} = 66.045$. Therefore, only *Run* 33 survives for future consideration.

Although *Run* 33 (the run with a fine-tuned option) is statistically "better" than *Run* 20 (the run without a fine-tuned option), care must be taken when using the fine-tuned option. In fact, *Run* 20 has the second minimum rank sum. Statistically speaking, it is not "worse" than *Runs* $31, 35$ and 36, and is consistently "better" than *Runs* 32 and 34. This means that the fine-tuned option is not necessarily always better than the simple base probability option. A significant amount of effort is required to find the best β_1 and β_2 values when using the fine-tuned option. For simplicity and robustness, the option without fine tuning has much to commend it.

Recovery Strategy. We now study the performance of various recovery strategies designated as A, B, C, and D.

Strategy A — the strategy used in the base algorithm.

Strategy B — This recovery process starts with the worst solution in the elite list and proceeds towards the best, as in *Strategy A*. However, the list is updated when the new solution is better than the worst of the remaining solutions. The new solution is inserted in an appropriate position on the list (according to its objective value) and the worst one is removed. After recovering from the best solution in the list, the search continues until either the termination condition is satisfied or a new best solution emerges as the next recovering solution.

Strategy C — This strategy starts from the best solution in the list and proceeds towards the worst (opposite to *Strategy A* and *B*). The list is updated

the same way as in *Strategy B*. After recovering from the worst solution in the list, the search continues until the termination condition is satisfied.

Strategy D — This is the "no restart" strategy. The search terminates when the termination condition is satisfied.

The rank sums of the tests are listed in Table 6. Note that the test for *Strategy A* duplicates *Run* 33 of the previous level.

Run	33	37	38	39
Strategy	A	B	C	D
$R_{i.}$	34.5	46.5	48	61

Table 6: Tests on Recovery Strategy

In Table 6, $f_r = 11.132 > \chi^2_{0.1,3} = 6.251$. The effects produced by various recovery strategies are evident at the confidence level 0.9. We prune *Runs* $37, 38$ and 39 because their rank sums are greater than $R_{33.} + z_\alpha * \sqrt{JI(I+1)/6} = 44.680$. Furthermore, *Strategy D* (no-restart strategy) is significantly worse than any recovery strategy, indicating that the recovery strategy is a very effective intensification strategy for the TS algorithm in this problem domain.

Frequency of Performing Swap Moves. Our TS algorithm oscillates between the two elementary (constructive and destructive) moves and the swap moves. The swap moves are performed either once in every n_1 iterations (for periodic perturbation), or performed consecutively n_2 times (for conditional oscillation) if the search cannot improve the solution for the past 200 iterations. We set

$n_1 = 7$ and $n_2 = 5$ in the base algorithm (as in *Run* 33). We attempt to find the best settings for n_1 and n_2 by comparing the six different settings as shown in Table 7.

Run	33	40	41	42	43	44
n_1	7	7	4	4	10	10
n_2	5	10	5	10	5	3
$R_{i.}$	39.0	70.5	77.5	60.0	74.0	78.0

Table 7: Tests on Frequency of Performing the Swap Moves

Table 7 shows the apparent dominance of *Run* 33 over the other five settings. Friedman's test statistic gives $f_r = 16.902 > \chi^2_{0.1,5} = 9.236$. We prune all *Runs* 40 through 44 since their rank sums exceed the critical value of $R_{33.} + z_\alpha * \sqrt{JI(I+1)/6} = 53.41$. Using the medium frequency of performing swaps in this situation works better than the other choices.

The foregoing search procedure for the best factor-treatments configuration is depicted in Figure 1. The number at each node in the search tree represents the run number, the "X" on a branch indicates that its leaf node is pruned by Friedman's Test while the "XX" means that the leaf node is pruned by the Wilcoxon test.

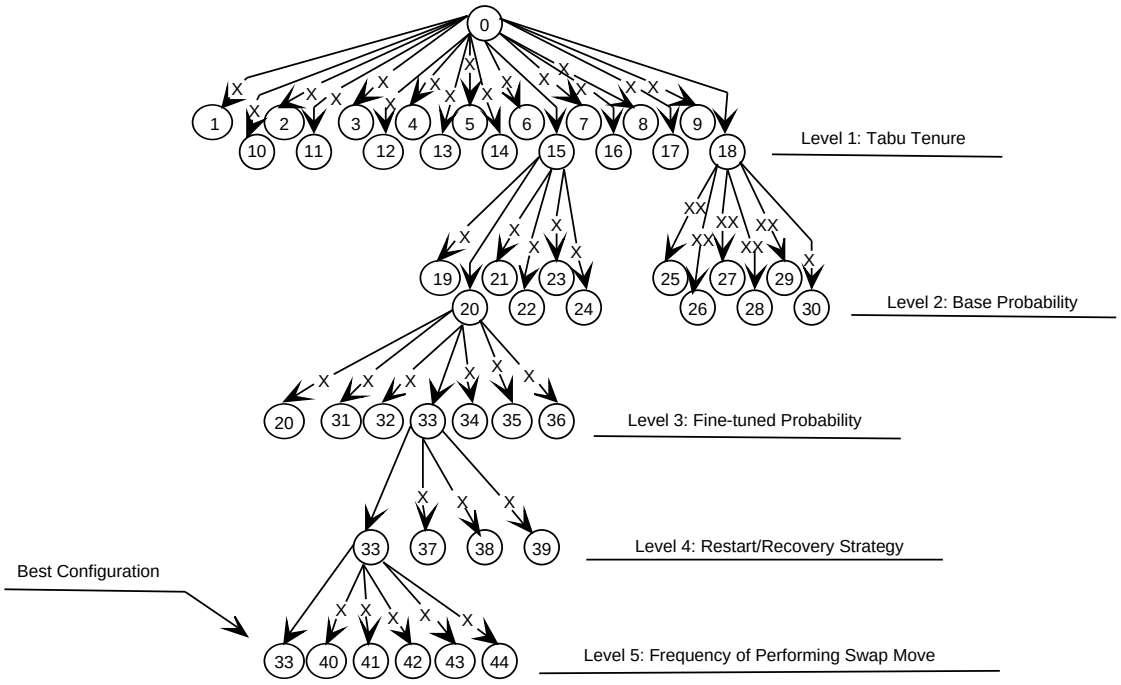

Figure 4.1: Search Tree of the Fine-Tuning Procedure

In Figure 1, only 44 runs in total are required, as compared to a total of 18144 runs required by complete enumeration. Although the time saving is enormous in this case, the degree of this saving depends on the power of the statistical tests to prune nodes, which is in turn constrained by the confidence level $1 - \alpha$. More runs may be required if a smaller α is used. In this study, we choose $\alpha = 0.1$ as a good compromise for pitting accuracy against computational expense.

Finally, we report the improvement achieved by our fine-tuned algorithm

over the base algorithm. Table 8 lists the objective function values of the nineteen test problems for both the base algorithm (*Run* 0) and the best improved algorithm (*Run* 33). For comparative purposes, we also list the best objective function value for each test problem over the 44 runs as the best upper bound.

Test Problem ($n \times m$)	Basic Algorithm Solution (Run 0)	Best Algorithm Solution (Run 33)	Percentage of Run 33 to Run 0	Best Upper Bound	Percentage of Run 33 to Best Upper Bound
(50×50)	50712	50712	100.00	50712	100.00
(100×125)	138522	109021	78.70	108940	100.07
(125×100)	52432	52432	100.00	52007	100.82
(100×300)	80633	72146	89.47	71958	100.26
(150×250)	178537	168017	94.11	167384	100.38
(200×200)	318339	281810	88.53	278054	101.35
(250×150)	273318	266037	97.34	265966	100.03
(300×100)	549320	549320	100.00	549320	100.00
(125×500)	214853	196238	91.34	192389	102.00
(225×400)	407349	378542	92.93	378542	100.00
(325×300)	666542	628757	94.33	623406	100.86
(425×200)	818524	794722	97.09	794275	100.06
(175×450)	275923	256531	92.97	252119	101.75
(275×350)	409245	389175	95.10	379036	102.67
(375×250)	206567	205268	99.37	203131	101.05
(475×150)	2067345	2067345	100.00	100.00	100.00
(400×500)	858599	869444	101.26	856727	101.48
(500×400)	653521	651271	99.66	644783	101.01
(450×450)	792166	751228	94.83	747719	100.47
Average	—	—	95.11	—	100.75

Table 8: Overall Performance of the Fine-tuned Algorithm

Table 8 shows that *Run* 33, the selected fine-tuned algorithm, significantly improves upon the base algorithm. For the nineteen test problems, it yields improved solutions in fourteen cases, tied solutions in four cases and a slightly worse solution in only one case. The average percentage improvement over the nineteen problems is 4.89%. Given the fact that our base algorithm already

yields very good solutions as shown in [9], the magnitude of the improvement obtained is noteworthy. (Such a difference can translate into significant cost savings in practical applications.) Furthermore, in comparison with the best upper bound, *Run* 33 yields "the-best-of-all" solutions in three cases. All solutions obtained by *Run* 33 are on average only 0.75% worse than the best upper bounds.

5. Conclusion

In this paper, we propose a systematic procedure to fine-tune a TS algorithm, employing the statistical tests to eliminate settings which prove to be inferior. We then conduct a computational study of this procedure to fine-tune a specific algorithm (already shown to be highly successful) for solving a telecommunications network design problem. The outcomes of this study lead to the following conclusions: (1) tabu tenures based on dynamic relative neighborhood sizes work better than those generated from a fixed interval, and larger tabu tenures should be assigned to discourage nodes from being transferred from the larger set to the smaller set; (2) a probabilistic TS choice rule consistently outperforms a deterministic choice rule and the base probability $p = 0.28$ is the best in this study; (3) the base probability can be further fine-tuned by linking it to the "goodness" of the candidate moves, but the two parameters involved

must be carefully selected; (4) a recovery strategy is a very effective intensification strategy, and a circular elite solution list produced the best form of such a strategy in our study; (5) the performance of our TS algorithm is influenced by the frequency of performing swap moves that are used for the perturbation and oscillation purposes, and the medium frequency proves superior to the other choices for performing these swap moves.

The statistically fine-tuned algorithm yields strictly improved solutions for over 73% (14 out of 19) of the test problems by comparison to the base algorithm which was previously the best known heuristic. In only one instance the new version obtains a (very slightly) worse solution than the base algorithm. We conclude that our fine-tuning approach using statistical tests can be a useful procedure for enhancing algorithmic design.

References

[1] Barr, R. S., B.L. Golden, J.P. Kelly, M.G.C. Resende and W.R.. Steart, 1995, Designing and Reporting on Computational Experiments with Heuristic Methods, *Journal of Heuristics*, **Vol. 1**

[2] Devore, J. L., 1991, Probability and Statistics for Engineering and the Sciences, Third Edition, Brooks/Cole Publishing Company, Pacific Grove, California.

[3] Glover, F., 1995, Tabu Search Fundamentals and Uses, *Working Paper*, Graduate School of Business, UNiversity of Colorado at Boulder, Boulder, CO.

[4] Glover, F. and M. Laguna, 1993, Tabu Search, in: C. Reeves, (eds.), *Modern Heuristics for Combinatorial Problems*, Blackwell Scientific Publishing.

[5] Glover, F. and A. Løkketangen, 1994, Probabilistic Tabu Search for Zero-One Mixed Integer Programming Problems, *Working Paper*, Graduate School of Business, University of Colorado at Boulder, Boulder, CO.

[6] Hetimansperger, T. P., 1984, Statistical Inference Based on Ranks, John Wiley & Sons, Inc.

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

[8] Mead, R., 1988, The Design of Experiments: Statistical Principles for Practical Applications, Cambridge University Press, Cambridge, UK.

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

Appendix

In this appendix, we formulate the STS problem as a 0-1 integer programming problem as follows. First we define:

T : set of target nodes;
S : set of Steiner nodes;
c_{ij} : cost of connecting target node i to Steiner node j;
d_{jk} : cost of connecting Steiner nodes j and k;
b_j : cost of activating Steiner node j.

The decision variables of this formualtion 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;
z_j : a binary variable equal to 1 if and only if Steiner node j is selected to be *active*.

The model is then

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

subject to:

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

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

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

$$\sum_{j \in S} \sum_{\substack{k > j \\ k \in S}} y_{jk} = \sum_{j \in S} z_j - 1, \tag{5}$$

$$\sum_{j \in R} \sum_{\substack{k \in R \\ k > j}} y_{jk} \leq \sum_{j \in \{R - w\}} z_j, \quad w \in R, \quad R \subset S, \; |S| \geq 3, \tag{6}$$

$$x_{ij} \in \{0, 1\}, \qquad i \in T, \quad j \in S, \tag{7}$$

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

$$z_j \in \{0, 1\} \qquad j \in S. \tag{9}$$

In this formulation, the objective function (1) seeks to minimize the sums of the connection cost between target nodes and Steiner nodes, the connection cost between Steiner nodes, and the setup cost for activating Steiner nodes. Constraint (2) specifies the star topology such that each target node must be connected to exactly one Steiner node. Constraint (3) indicates that the target node can only be connected to the active Steiner node. Constraint (4) stipulates that two Steiner nodes can be connected if and only if both nodes are active. Constraints (5) and (6) express the spanning tree structure over the active Steiner nodes. In particular, (5) speicifies the condition such that the number of edges in a spanning tree equals one less than the number of

nodes, while (6) is an *anti-cycle* constraint that also compels the connectivity for each active Steiner node via the spanning tree. Constraints (7), (8) and (9) state the nonnegativity and discrete requirements for the variables. All decision variables are defined as binary.