Theory and Methodology

An in-depth empirical investigation of non-greedy approaches for the minimum spanning tree problem

F. Glover
School of Business, University of Colorado, Boulder, CO 80309-0419, USA

D. Klingman
Graduate School of Business, and Computer Sciences, College of Natural Sciences, The University of Texas at Austin, Austin, TX 78712, USA

R. Krishnan
Decision Systems Research Institute, School of Urban and Public Affairs, Carnegie-Mellon University, Pittsburgh, PA 15213, USA

R. Padman
School of Urban and Public Affairs, Carnegie-Mellon University, Pittsburgh, PA 15213, USA

Received March 1989; revised September 1989

Abstract: This paper details the design, implementation, and testing of a one edge pass non-greedy algorithm for the minimum spanning tree problem. The use of network labeling procedures, and path weights to screen entering edges, provides a novel implementation of this algorithm. The choice of data structures also allows for an efficient implementation of reoptimization procedures, which have hitherto never been studied. Comprehensive results on the performance of both the greedy and non-greedy algorithms in optimization and reoptimization modes under various screening criteria, graph topologies, sizes and densities, and weight distributions are presented. The empirical results based on 855 problems bear out Tarjan's conjecture that greedy algorithms have a competitive advantage over non-greedy algorithms for the minimum spanning tree problem, except in special cases. However, for the problems tested, when a small percentage of edge weights are changed, non-greedy approaches are competitive with greedy algorithms in reoptimization mode.

Keywords: Networks, minimum spanning tree, non-greedy approaches, reorganisation

1. Introduction

The minimum spanning tree (MST) problem is one of the simplest and most widely studied graph problems (Cheriton and Tarjan, 1976, Dijkstra, 1959, Graham and Hell, 1985, and Kershenbaum and Van Slyke, 1972). The MST problem is useful in the modeling of a variety of applications such as the design of cable television networks, design of leased-line telephone networks, and other telecommunications problems (Loberman and Weinberger, 1957). It has also found important applications in cluster analysis by providing efficient modeling techniques to solve problems that have not been handled well by clustering methods (Gower and Ross, 1969), and in network reliability, where the weight of a MST represents the minimum probability that the tree will fail at one or more edges (Van Slyke and Frank, 1972). Gomory and Hu (1961) have used MST evaluations as subproblems for solving multiterminal flow problems. A similar application was proposed by Held and Karp (1970, 1971) for solving traveling salesman problems. Applications to solving specialized transportation networks (Golden, Magnanti, and Nguyen, 1977), and to suboptimizing or decomposing larger, more complex problems further illustrates the pervasive nature of the MST problem.

The MST problem is one of the few problems that can be solved via a 'greedy approach' (Edmonds, 1971). There are three classical algorithms which follow this approach, namely, those of Boruvka (1926) (also rediscovered by Kruskal, 1956), Choquet (1938) (also rediscovered by Sollin (Berge and Ghouila-Houri, 1965)), and Prim (1957). Numerous greedy variants of these basic algorithms have been proposed in the literature (Pynn and Warren, 1972, Haymond, Jarvis and Shier, 1984, Kevin and Whitney, 1972, Yao, 1975, and Cheriton and Tarjan, 1976). Cheriton and Tarjan (1976) have performed an extensive study of the theoretical efficiency of MST algorithms, and Kershenbaum and Van Slyke (1972) provide an excellent survey of implementation techniques for such algorithms. Efficient implementation techniques and extensive computational testing of these algorithms have also been documented in the literature (Brennan, 1982, Haymond, Jarvis, and Shier, 1984, Jarvis and Whited, 1983, Johnson, 1975, Kershenbaum and Van Slyke, 1972, and Spira and Pan, 1975).

MST greedy algorithms are analogous to label-setting shortest path algorithms. *Label-setting* procedures for shortest path problems (Gilsinn and Witzgall, 1973) dictate that once an arc has been selected for inclusion in the tree, it is never discarded. Hence the tree is optimal at every stage with respect to the arcs selected. Greedy algorithms follow the same approach. On the other hand, *label-correcting* shortest path methods may be interpreted as performing exchanges that swap an edge in the tree with an edge out of the tree to progressively improve the solution. Label-correcting approaches have been shown to be superior to label-setting approaches in the study of shortest path problems over a wide range of topologies (Glover, Klingman, Phillips, and Schneider, 1985).

Non-greedy MST approaches are the analog of label-correcting methods for the shortest path problem. There have been some conjectures regarding the efficiency of such non-greedy approaches for the MST problem (Tarjan, 1983), but no empirical study exists in the literature to verify them. The research study of this paper undertakes to remedy this lack. As a foundation for this study, we develop non-greedy algorithm variants based on efficient network flow labeling procedures, and conduct in-depth empirical tests comparing these non-greedy algorithms to greedy algorithms for the MST problem.

Another motivation for studying non-greedy approaches lies in their relevance to the design of reoptimization procedures for the minimum spanning tree problem. Surprisingly, no study of MST reoptimization procedures has been conducted to date. The ability of non-greedy algorithms to start from any initial spanning tree, in much the same manner as label-correcting shortest path algorithms, suggests that non-greedy algorithms may be a natural vehicle for reoptimization when only a few of the edge weights change. Moreover, in applications which employ MST problems as subproblems, the edge weights often change from one subproblem to the next, requiring efficient techniques for obtaining the new solution. Such related subproblems can be solved either by starting fresh or by employing knowledge about the old optimal MST. This occurs in the solution of multiterminal network flow problems, traveling salesman problems, and the more general degree-constrained minimum spanning tree problems (Gomory and Hu, 1961, Gabow and Tarjan, 1985, Glover and Klingman, 1974, Glover and Novick, 1986, and Held and Karp, 1970).

2. Notation and terminology

Let $G = (N, E)$ denote a connected undirected graph with node set N and edge set E.

Each edge $e=(i, j)$ in E has an associated weight $w(e)=w(i, j)$, unrestricted in sign. The weight $w(T)$ of a spanning tree $T=(N, E')$ is given by

$$w(T)=\sum_{e \in E'} w(e).$$

A *minimum spanning tree* (MST) for G is then a spanning tree T' such that $w(T') \leqslant w(T)$ for all spanning trees T of G.

The data structures commonly used to represent and manipulate rooted spanning trees are briefly reviewed as follows. (For a complete description, see Glover, Karney, and Klingman, 1972, and Barr, Glover, and Klingman, 1979.) If nodes i and j denote the end points of a common edge in the rooted tree such that node i is closest to the root, then node i is called the *predecessor* of node j, and j is called the *immediate successor* of node i. The *cardinality function* defines for each node k the number of nodes in its subtree. The *thread function* defines a top-to-bottom, left-to-right node pointer through the tree. All immediate descendants of a node k are called *brothers* of each other. The left-most node among these is identified by the *successor function* of node k, and the remaining are identified by the *brother function*. The *lastnode function* of any node k is the last node encountered in the subtree of k when the subtree is traversed in thread order.

For any edge $e=(i, j)$ in $E-E'$, where E' is the set of edges forming the tree, the *basis equivalent path*, $\mathrm{PT}(e)$ of e, is defined as the set of tree edges in the unique cycle created by the addition of e to E'.

In this paper, we create and utilize a new node label, called the *path-weight* label, which is an upper bound on the maximum edge weight on the unique path from the node to the root. This label is used to improve computational efficiency of the non-greedy algorithm variants.

3. Non-greedy algorithm for the MST problem

The following optimality condition characterizes minimum spanning trees (Aho, Hopcroft, and Ullman, 1974, Haymond, Jarvis, and Shier, 1984, and Tarjan, 1983):

Suppose T is a spanning tree of G. T is an MST if and only if for all edges e not in T, $w(e)>w(u)$ for all edges u in $\mathrm{PT}(e)$.

Haymond et al. (1984) observe that this optimality condition immediately suggests the following algorithm. Construct an arbitrary spanning tree for G, and test whether the optimality condition holds. If so, the current tree is optimal. Otherwise, there are edges e and u such that the condition is not satisfied, and it is therefore advantageous to exchange e and u in the current tree. Henceforth we will call such an exchange, an *advantageous exchange*. By repeating this test/exchange procedure, an MST is obtained after a finite number of steps.

If e is selected arbitrarily, we refer to such an algorithm as *non-greedy*. By the following rule to select u, the number of times an edge has to be examined is reduced to one.

Leaving Edge Selection Criterion. For any-advantageous exchange (e, u), select u such that $w(u)=\operatorname{maximum}_{i \in \mathrm{PT}(e)}(w(i))$.

Lemma: *A non-greedy algorithm which employs the Leaving Edge Selection Criterion has to examine each edge not in the starting tree exactly once.*

Proof: There are two cases: (1) e is in the starting tree, and (2) it is not. First, assume e was in the starting tree and subsequently removed but never examined. In this case, some edge r replaced e, and $w(e)$ was the largest weight in $\mathrm{PT}(r)$. Suppose immediately after the exchange (r, e), e had been examined. Its basis equivalent path would consist of the edges $(r \cup \mathrm{PT}(r)-e)$. Since $w(e)>w(i)$, $i \in r \cup \mathrm{PT}(r)-e$, e would not provide an improving exchange at this time. Further, the same argument establishes that any edge e which leaves the tree will not provide an improving exchange if considered immediately after leaving the tree.

We will now show that an edge which is not improving when it is considered will never provide an improving exchange. Let e be an arbitrary edge in the graph and $\mathrm{PT}(e)$ denote the basis equivalent path of e when it was examined. $\mathrm{PT}(e)$ will change only if an advantageous exchange (x, y) occurs such that $y \in \mathrm{PT}(e)$. Suppose such an exchange occurs and the Leaving Edge Selection Criterion is applied. Let $\mathrm{PT}(x)$ denote the basis equivalent path of x at the time of the exchange. Then we have:

(1) $w(e)>w(y)$ because e was not improving when it was examined,

(2) $w(y) > w(x)$ because (x, y) is an advantageous exchange, and

(3) $w(y) \geqslant w(i)$ for $i \in \text{PT}(x)$ because the Leaving Edge Selection Criterion was applied. Since the basis equivalent path of e after the exchange (x, y) is a subset of $\text{PT}(e) \cup \text{PT}(x)$ and since $w(e) > w(i)$, $i \in \text{PT}(e) \cup \text{PT}(x)$, e would not provide an improving exchange after performing any improving exchange which altered the basis equivalent path of e. □

The first step of a non-greedy algorithm is to obtain an initial spanning tree. The starting tree may consist of edges in E, in which case it will be called a *feasible spanning tree*. Otherwise, the tree consists of one or more edges not in E. Such edges will be referred to as *artificial edges* and assigned a weight greater than the largest edge weight of the edges in E. A tree containing one or more artificial edges will be referred to as an *artificial spanning tree*.

The basic steps of a one edge pass non-greedy algorithm follow.

Algorithm BNG (Basic Non-Greedy Algorithm).

Step 1. Build an initial spanning tree $T = (N, E')$, possibly containing artificial edges whose edge weights are larger than the maximum edge weight in E. Select an edge $e = (i, j)$ in $E - E'$.

Step 2. Test the optimality condition; if T is non-optimal, select u based on the Leaving Edge Selection Criterion. Otherwise, go to Step 4.

Step 3. Perform the exchange (e, u), updating T (and hence E).

Step 4. Select an edge not previously examined from the set $E - E'$, where E' is the set of edges in the starting tree, and return to Step 2. If all edges have been examined, stop. If the current spanning tree T contains an artificial edge, the problem is infeasible. Otherwise, it is an MST.

Justification of the algorithm follows from the preceding lemma.

Since Step 2 of BNG has a worst case bound of $O(n)$, the algorithm has a worst case bound of $O(mn)$, where m is the number of edges in E. We propose an enhancement which can reduce the number of basis equivalent path cycle traces performed in Step 2 and thus may improve the empirical run time of the basic algorithm. The result is termed the *Intelligent Non-Greedy (ING) algorithm*.

Specifically, the ING algorithm employs the path weight function to eliminate unnecessary cycle traces. Recall that a path weight is defined to be an upper bound on the maximum edge weight of the unique path from the given node to the root node. Thus, for every edge $e = (i, j)$ considered in Step 2 of the basic algorithm, if $w(e)$ is greater than or equal to Path-weight(i) and Path-weight(j), then no advantageous exchange exists for e. The path weight function, however, provides only partial screening in the sense that, for some edges, a comparison of $w(e)$ with its associated path weights does not preclude a cycle trace which may not result in an advantageous exchange.

The basic steps of the ING algorithm are the same as the BNG algorithm except that Step 2 is replaced by Step 2a.

Step 2a. If $w(e) >$ Path-weight(i) and $w(e) >$ Path-weight(j), discard edge e; else trace PT(e), identify u, the maximum weight edge in PT(e); if $w(u) < w(e)$, discard edge e, and go to Step 4.

The worst case complexity of the ING algorithm is the same as the BNG algorithm, $O(mn)$.

4. Implementation issues

The implementation of the one edge pass non-greedy algorithm provides several opportunities for utilizing network flow tree data structures. Several versions of both the BNG and ING algorithms were developed, based on different start procedures, alternative data structures, and path weight functions of varying restrictiveness.

The basic computational tasks involved in implementing the non-greedy algorithms are in creating the initial tree, performing cycle traces, and updating the node label functions. Each of these tasks may be implemented using several alternative strategies. This section considers the alternative starting procedures and data structures used and presents a brief discussion of the trade-offs.

Two approaches were examined for building the initial tree. One of these approaches creates a starting tree composed of only artificial edges, and the other approach creates a starting tree composed of no artificial edges. The artificial start procedure, called AA, creates an artificial node and links all nodes in the graph to it using artificial edges with an edge weight set equal to 1 unit greater than the largest edge weight in the graph.

To construct a feasible spanning tree, we chose to reorder the edge list, in what is often called *forward star form*, so that all edges incident to a node are located contiguously. A breadth-first tree is then generated using this reordered list, or the problem is determined to be infeasible. This start procedure will be called RR.

The data structures used to implement the cycle trace and update procedures are presented in increasing order of complexity. The basic algorithm BNG was implemented using only the predecessor and cardinality functions. In Step 2, for a given edge e, its end nodes are identified, and their predecessors are traversed using the cardinality function (to ensure that only edges in PT(e) are examined) until a common ancestor is encountered. This implementation is efficient in the sense that it never considers any edge not in PT(e) and thus simultaneously allows the edge u in PT(e) with the largest weight to be identified. If $w(u) > w(e)$, an advantageous exchange is performed and the predecessor and cardinality functions are simultaneously updated. Another advantage of this approach is that the cardinality function only has to be updated for the nodes in PT(e); whereas other functions which could be used in place of cardinality require a subtree update, e.g., the depth function.

To reduce the number of cycles traces, the path weight labels are employed in ING. In the ING implementation, the cycle trace is still performed using cardinality and predecessor functions, but the use and updating of the path weight labels necessitates additional consideration.

Thus to thoroughly evaluate the effectiveness of the path weight concepts and the trade-off of the effort required to update them versus reducing the number of cycle traces, we looked at alternative strategies on the tightness of the path weight bounds and on when the algorithm begins to maintain and utilize the path weight function.

More specifically, the use of the path weight labels necessitates an update of these labels after each advantageous exchange. This update can be implemented as either a partial (P) or full (F) update. A partial update of the path weight function creates *sharp* path weights (i.e., path weights which are minimum upper bounds) on the nodes involved in the cycle trace alone. It may leave some of the remaining nodes in the tree with *non-sharp* values (i.e., path weights which are upper bounds, but not necessarily minimal). This procedure can be performed effectively using the predecessor, cardinality, and path weight functions and only involves a partial retrace of PT(e).

In contrast, a full update maintains sharp path weights for all the nodes of the tree at all times. This can be done efficiently by considering only the nodes in the subtree created when the largest weight edge $u = (i, j)$ leaves the spanning tree. Let k denote the node i or j whose predecessor is j or i, respectively. The subtree of node k is rehung using the new edge e whose weight is less than that of the leaving edge. Thus, the only nodes whose path weight will not be sharp after the exchange lie in subtree of k. Consequently, two different sets of data structures were examined to efficiently traverse and update path weights in subtree of k. One implementation uses the predecessor, thread, and lastnode functions to traverse and update path weights in the subtree. The other implementation uses the successor and brother functions to maintain sharp path weights.

Given the effort required to maintain the path weight labels and the possibility that the path weights may be particularly ineffective at eliminating cycle traces when the current spanning tree contains artificial edges, we also made an evaluation as to when an algorithm should begin to maintain and use path weight labels. Specifically, when the AA start is employed, path weight labels are not maintained until a feasible tree is found. At this point, the algorithm initializes and updates path weight labels using either the partial or full update procedure.

Four variants of the non-greedy algorithm were implemented based on the above discussion. The first variant, called RR, is an implementation of algorithm BNG. It uses the predecessor and cardinality functions. The second variant, called RR-P, is an implementation of algorithm ING with partial updates of the path weights using predecessor, cardinality and path weight functions. The third, called RR-F-TL, and fourth, called RR-F-SB, variants are also implementations of the ING algorithm which employ complete updates; the former uses predecessor, cardinality, path weight, thread, and lastnode functions, while the latter uses predecessor, cardinality, path weight, successor, and brother functions. These variants permit a thorough comparison of path weight function alternatives, as well as performance of each data structure in the update procedure. These variants are then tested with the two start procedures, thus resulting in two BNG codes and six ING codes. The codes employing the nonartificial start have the above names. The codes using an artificial start have the same names, except that the first two letters, RR, are replaced by AA.

5. Experimental design

Eight variants of the non-greedy algorithm were developed and tested on a variety of problem sizes, densities, topologies, and weight distributions, and the best among them were compared with two implementations of Prim's algorithm, called P1 and P2, and one of Kruskal's, called K1. Table 1 specifies the memory requirements of each code. The best ING code from a memory standpoint is AA-P. Implementation and testing of these greedy-algorithm-based codes are reported in Haymond, Jarvis, and Shier (1984). P1 and K1 use a heap sort to select the least weight edge to be included in the tree, and P2 uses an address calculation sort to select the next eligible edge.

Comparative performance of alternative implementation strategies is studied in this paper by solving 855 randomly generated problems. All codes are written in FORTRAN and were executed on the IBM 3081D under optimization level 2. Five test problems were solved for each problem size, starting with a different random number seed, and the average solution time is reported. The reported times exclude input and output time. In order to ensure fair and consistent comparisons across all algorithms, a standard problem representation consisting of problem data in random edge format is input to all algorithms. This was selected as a standard because, traditionally, undirected graphs have been represented as edge lists (Aho, Hopcroft, and Ullman, 1974). Furthermore, Prim codes and some ING codes would have an undue advantage over the other codes if data is presented in a different format to these algorithms, such as forward star form. Thus the total time includes the time required to convert this standard representation of random edge format into forward star form for those algorithms that require it, such as Prim's codes, and those ING codes that use a non-artificial start.

For all test problems, edge weights are within the range 1 to 2000. Computational testing includes edge weights generated from two distributions – the uniform probability distribution and the beta distribution. In addition, four skewness levels were investigated for the beta distribution the first with 75% of the weights skewed to the right, the second with 75% of the weights skewed to the left, the third with 60% of the weights skewed to the right, and the fourth with 60% skewed to the left. This appears to be the first testing of non-uniform edge weights. Fifteen problem sizes were tested, ranging from 100 nodes and 200 edges to 2000 nodes and 20000 edges. For each node size, problems were generated with node degrees of 2, 5, and 10, where node degree is computed as the ratio of number of edges to number of nodes in the graph.

Test problems from three distinct graph topologies were generated. The first topology consists of random graphs, where two nodes are selected randomly to form a new edge to add to the graph. This graph generation code was obtained from Haymond, Jarvis, and Shier (1984). A uniform probability distribution is used to select the nodes.

The second topology is called a *bi-clustered* graph. These problems are generated by dividing the problem size into two equal groups and gen-

Table 1

Code	K	P1	P2	AA	RR	AA-P	RR-P	AA-F-SB	AA-F-TL	RR-F-SB	RR-F-TL
Node arrays	4	5	5	3	4	4	5	6	6	7	7
Edge arrays	5	5	5	3	6	3	6	3	3	6	6

erating distinct random graphs for each group. Two nodes are then randomly selected from the two disjoint graphs and connected with an edge having a weight of 2000.

The third topology, called a *single-clustered* graph, denoted as S-clustered in the tables of solution time, again consists of two groups, where one group consists of a single node connected to a randomly selected node from the second group with an edge weight of 2000. The single node is itself selected randomly from the given set of nodes. Bi-clustered and single-clustered graphs force all algorithms, irrespective of greedy or non-greedy structure, to scan all edges, thus providing a measure of the worst-case empirical performance of these algorithms. This also appears to be the first testing of MST algorithms on non-random graph topologies.

6. Computational results

After solving the 855 test problems and analyzing the results for the eight non-greedy codes, we found that the major conclusions regarding their relative performance could be expressed in one summary table, Table 2. Table 2 contains the aggregate solution time in milliseconds across all weight distributions, but distinguishes the problems by topology and node degree.

The results indicate that the AA-P code is the most robust. It dominates other non-greedy implementations across alternative topologies and in problems with node degrees of five and ten. However, the code RR proves better for problems of node degree two. The success of AA-P is due in part to two factors. First, its artificial start allows it to build a start tree quickly. Further, no path-weight labels are calculated or updated until all artificial edges are removed. At this point, the tree so obtained is typically a good tree resulting in quality path-weight labels. These labels help it to screen a large number of remaining non-tree edges quickly. It is interesting to observe that the behavior of AA-P is a function of both problem topology and node degree. While solution times are directly proportional to node degree, AA-P performs better in random and S-clustered topologies than in the bi-clustered topology. This is due to the special structure of the bi-clustered topology (two clusters linked by a single edge) which yields trees comprised of two subtrees derived from each cluster linked by a single edge which is always part of the tree. Exchanges performed on this structure in AA-P often alter the

Table 2
Aggregate solution times (milliseconds) for variants of the non-greedy algorithm on the IBM 3081D

Codes	RR-F-SB	RR-F-TL	RR-P	RR	AA-F-SB	AA-F-TL	AA-P	AA	Best
Topology. random									
Node degree. 2	717	745	552	537	687	676	649	586	RR
Node degree 5	1572	1607	1587	1748	1498	1645	1349	1724	AA-P
Node degree: 10	2260	2469	2966	3656	2483	2502	2088	3328	AA-P
Subtotal	4549	4821	5105	5941	4668	4823	4086	5638	AA-P
Topology bi-clustered									
Node degree. 2	738	754	498	464	576	564	540	481	RR
Node degree 5	1676	1767	1441	1449	1462	1479	1341	1347	AA-P
Node degree 10	2900	3040	2472	2986	2674	2581	2396	2663	AA-P
Subtotal	5314	5561	4411	4899	4712	4624	4277	4491	AA-P
Topology S-clustered									
Node degree 2	761	787	558	535	682	671	645	586	RR
Node degree 5	1552	1675	1541	1733	1664	1646	1443	1664	AA-P
Node degree. 10	2308	2494	2952	3590	2510	2467	2099	3323	AA-P
Subtotal	4621	4956	5051	5858	4856	4784	4187	5573	AA-P
Subtotal – Node degree. 2	2216	2286	1608	1536	1945	1911	1834	1653	RR
Subtotal – Node degree 5	4800	5049	4569	4930	4624	4770	4133	4735	AA-P
Subtotal – Node degree· 10	7468	8003	8390	10232	7667	7550	6583	9314	AA-P
Grand Total	14484	15338	14567	16698	14236	14231	12550	15702	AA-P

root path of several nodes resulting in low-quality path-weight labels which in turn reduce efficiency.

It is important to note that AA-F-TL and AA-F-SB, which implement full updates of the path-weight label, do not perform as well as AA-P even though they employ the same start procedure. This is due to the fact that the work used to maintain the quality of the path-weight labels dominates gains made in screening out non-tree

Table 3
Aggregate solution times (milliseconds) on the IBM 3081D for the skewed weight distributions

Codes	AA-P	K	P1	P2	Best
Topology. random; Node degree 2					
Right skewed	1125	827	854	627	P2
Left skewed	1339	821	858	907	K
Subtotal	2464	1648	1712	1534	P2
Topology random, Node degree: 5					
Right skewed	2604	1479	1349	994	P2
Left skewed	2749	1470	1336	1270	P2
Subtotal	5353	2949	2685	2264	P2
Topology random, Node degree 10					
Right skewed	4172	2277	2059	1699	P2
Left skewed	3950	2323	2078	1947	P2
Subtotal	8122	4600	4137	3646	P2
Topology bi-clustered, Node degree: 2					
Right skewed	1138	827	785	617	P2
Left skewed	1303	828	788	907	P1
Subtotal	2441	1655	1573	1524	P2
Topology bi-clustered, Node degree 5					
Right skewed	3020	1501	1286	992	P2
Left skewed	2981	1499	1312	1269	P2
Subtotal	6001	3000	2598	2261	P2
Topology. bi-clustered, Node degree 10					
Right skewed	5049	2239	1999	1790	P2
Left skewed	5239	2266	2010	1846	P2
Subtotal	10288	4505	4009	3636	P2
Topology S-clustered, Node degree 2					
Right skewed	1200	821	846	612	P2
Left skewed	1473	830	854	924	K
Subtotal	2673	1651	1700	1536	P2
Topology S-clustered, Node degree· 5					
Right skewed	2508	1485	1349	1055	P2
Left skewed	2696	1516	1346	1285	P2
Subtotal	5204	3001	2695	2340	P2
Topology S-clustered; Node degree 10					
Right skewed	4103	2276	2081	1710	P2
Left skewed	4274	2272	2094	1959	P2
Subtotal	8377	4548	4175	3669	P2
Grand total	50923	27557	25284	22410	P2

edges, resulting in less efficient solution times. The alternative data structures used in the update of the path-weight labels do not seem to make any significant difference.

In all our testing, the code RR performed well on problems with node degree two. Codes which employ a real start such as RR build a tree from edges in the problem data and have no artificial edges. Since $n - 1$ edges, where n is the number of nodes, are required to build a tree and since problems with node degree two have $2n$ edges, a real start procedure supplies a start tree with about half the available edges already in it. Thus, RR, even without a path-weight label, is efficient due to the small number of edges considered. However, as node degrees increase the performance of RR drops sharply.

In fact, as shown in Table 2, real start procedures in general seem to lose heavily in comparisons on the larger node degrees and topologies. Real starts provide a tree composed of real edges quickly. However, the quality of the tree is poor, resulting in poor path-weight labels. These low-quality labels are unable to screen a large proportion of non-tree edges, regardless of the use of a full or partial update procedure, and result in high solution times. Thus, our comparative testing of non-greedy implementations indicates that an artificial start combined with a partial update is the most efficient.

While AA-P is the best non-greedy code, it is completely dominated by our implementations of Prim's algorithm and Kruskal's algorithm. In fact, Prim is more than twice as fast as AA-P. These tests were conducted using problems whose edge weights were generated from-random and skewed distributions. Tables 3 and 4 show the-results of the testing on skewed and uniform distributions, respectively. P2, the implementation of Prim with the address calculation sort, proved to be both robust and efficient in all our testing. This directly bears out the superiority of greedy approaches over non-greedy approaches in solving the minimum spanning tree problem.

Several interesting observations can be made from the testing on skewed and uniform distributions. In the case of the greedy algorithms, the type of sorting procedure plays an important role in the effect of skewness on solution times. As can be seen in Table 3, the K and P1 implementations of Kruskal and Prim algorithms, which employ a heap sort, are unaffected by the weights generated from the skewed distribution. However, P2, the implementation of Prim's algorithm with an address calculation sort, runs faster when the bulk of the edge weights have smaller magnitudes. This is due to the fact that an address calculation sort locates edges in an array based on the magnitude of its edge weight. In the event that the bulk of the edge weights have large

Table 4
Aggregate solution times (milliseconds) on the IBM 3081D for the uniform weight distribution

Codes	AA-P	K	P1	P2	Best
Topology random					
Node degree 2	2596	1650	1706	1798	K
Node degree 5	5396	3033	2705	2236	P2
Node degree 10	8352	4592	4146	3495	P2
Subtotal	16344	9275	8557	7529	P2
Topology bi-clustered					
Node degree 2	2160	1661	1575	1790	P1
Node degree 5	5364	3026	2569	2202	P2
Node degree 10	9584	4599	3999	3458	P2
Subtotal	17108	9286	8143	7450	P2
Topology. S-clustered					
Node degree 2	2580	1650	1698	1770	K
Node degree 5	5772	3050	2726	2268	P2
Node degree 10	8396	4549	4141	3534	P2
Subtotal	16748	9249	8565	7572	P2
Grand total	50200	27810	25265	22551	P2

values, location of an edge in the array typically involves more searching resulting in slower solution times.

One final note seems appropriate concerning implementations of greedy algorithms. While previous empirical studies (Haymond et al., 1984) have concluded that the Kruskal approach is generally much slower than the Prim approach, our results do not indicate this to be true. It appears that the discrepancy may be due to our use of a standard problem format. In Haymond et al. (1984), the inputs are provided in the form required by the implementation, without counting any time used to organize the problem inputs in these forms. By contrast, our solution times include the effort to transform problem inputs from a randomly ordered collection of edges into the forms the various codes require.

AA-P also seems to be slightly affected by the skewed weight distributions and generally (though not exclusively) executes slower on problems with skewed edge weights than on problems with uniform weight distributions. However, it does not demonstrate, in any conclusive manner, improved efficiency on either left- or right-skewed edge-weight values.

In summary, AA-P proved to be the most efficient and robust non-greedy code while P2 was the most robust and efficient greedy code. Further, all greedy codes dominate the AA-P code by a factor of almost two.

7. Reoptimization

Solution procedures for several interesting problems such as the multiterminal flow problem, the traveling salesman problem and the vehicle routing problem often employ Lagrangean relaxation where the relaxed problem is an MST problem whose weights change from iteration to iteration. In such applications, the ability to obtain a new optimal MST solution quickly is very important.

Reoptimization may be carried out either by solving the modified problem anew or by starting from the previous optimal solution. The former strategy can be implemented by greedy approaches, but the latter strategy seems more suited to implementation by non-greedy approaches, which have the ability to start from an arbitrary tree. We were unable to find any previous work in the literature which addressed reoptimization issues for the MST problem.

To test reoptimization using a non-greedy procedure, we started from an MST denoted by $T(N,E',w(E'))$ for problem $G(N,E,w(E))$, and upon the introduction of a new weight function $\bar{w}(E)$, we simply used tree $T(N,E',\bar{w}(E'))$ as the starting tree for problem $G(N,E,\bar{w}(E))$. The procedure was implemented by providing the partial- and full-update artificial start codes with the tree $T(N,E',w(E'))$ and the edge weight changes. These codes are called MST-P and MST-F, respectively. The results were compared with the times to solve the modified problem using P2, since it was the best greedy algorithm. To test reoptimization, we chose to create two types of test scenarios. In the first, only the weight of edges not in the old MST were changed. In this case, each edge weight was decreased sufficiently to ensure that the old MST was no longer optimal. In the second, only the weight of edges in the old MST were changed. In this case, each edge weight was increased sufficiently to ensure that the old MST was no longer optimal. Each of these scenarios ensured that the non-greedy algorithm would have to perform several exchanges.

In the reoptimization testing, five problem sizes in the range 100 nodes to 2000 nodes with three node degree densities (2, 5, 10) and three problem topologies (random, bi-Clustered, S-Clustered) were tested. (Since the edge weights were selected to ensure non-optimality of the old MST, the weights can no longer be considered to be randomly selected from a given distribution. However, the original weights are from the same distributions as in the earlier testing.)

For each scenario, we tested changing 5, 10, 15, and 20 edge weights at a time. Table 5 presents the aggregate solution times over all topologies for the case where the weight of edges in the old MST are increased. Additionally, the first three columns in Table 5 specify the solution times according to the number of edge weights increased for edges in the old MST and the next three columns specify the solution times according to node degree 2, 5, and 10. Table 6 presents similar results for the scenario where the edge weights are decreased for edges not in the old MST. The tables contain the times for three codes: P2, the most efficient greedy algorithm, MST-P, the reoptimization algorithm with a partial update of the path-weight label, and the MST-F, the reoptimization algorithm employing a full update of the path-weight label.

As can be observed from the results in Table 5, MST-F dominates all codes at the four levels of changes, when edge weights are increased for edges in the old MST. The efficient performance of MST-F may be attributed to the fact that the old optimal tree provided quality path-weight labels, and the full update procedure (which maintains the quality of these labels) facilitated efficient screening of a large number of non-tree edges. The use of the full update seems crucial since MST-F substantially outperforms MST-P.

It is interesting to note from Table 5 that the time for reoptimization increases slowly with the number of weight changes in the case of the MST-F code. It increases at a faster rate in the MST-P code, and remains fairly constant in the case of P2 (where it is independent of the number of changes since P2 solves the problem anew every time). This clearly indicates that in performing reoptimization with a relatively small number of changes, MST-F has an advantage over other methods, but as the number of changes increases, P2 eventually dominates. Another point to observe is that the solution times are a function of the number of nodes for all codes when the number of changes is small. This property

Table 5
Aggregate solution times (milliseconds) on IBM 3081D for in-tree edge-weight changes

Nodes	Codes			Codes		
	MST-F	MST-P	P2	MST-F	MST-P	P2
	5 changes [a]			Node degree = 2		
100	39	74	57	46	39	84
300	108	231	160	102	115	202
700	211	508	379	196	275	497
1000	341	626	544	462	644	688
2000	782	2174	1153	644	718	1499
Total	1481	3613	2293	1450	1791	2970
	10 changes [a]			Node degree = 5		
100	50	88	60	79	107	71
300	131	258	158	183	128	217
700	279	1004	379	428	1229	461
1000	413	896	540	563	1375	662
2000	818	2341	1149	1089	2206	1220
Total	1691	4587	2286	2342	5045	2631
	15 changes [a]			Node degree = 10		
100	64	97	60	105	215	85
300	136	286	158	240	643	236
700	339	1231	382	581	2485	632
1000	452	907	545	659	1569	832
2000	1022	4092	1143	2056	10279	1719
Total	2013	6613	2288	3641	15191	3504
	20 changes [a]					
100	77	106	63			
300	157	339	162			
700	374	1281	413			
1000	478	1067	553			
2000	1127	4606	1144			
Total	2213	7397	2335			

[a] Number of edge-weight changes

Table 6
Aggregate solution times (milliseconds) on IBM 381D for non-tree edge weight changes

Nodes	Codes			Codes		
	MST-F	MST-P	P2	MST-F	MST-P	P2
	5 changes [a]			Node degree = 2		
100	30	24	58	35	28	79
300	79	68	155	68	58	196
700	182	161	380	148	129	489
1000	272	240	540	211	190	679
2000	597	546	1141	588	508	1484
Total	1160	1039	2274	1050	913	2925
	10 changes [a]			Node degree = 5		
100	31	27	58	40	33	69
300	84	71	155	106	88	188
700	187	163	382	231	203	427
1000	276	245	559	377	334	655
2000	618	551	1138	793	707	1169
Total	1196	1057	2292	1547	1365	2508
	15 changes [a]			Node degree = 10		
100	36	29	58	63	51	83
300	88	72	155	169	141	237
700	191	164	376	379	326	562
1000	283	251	540	535	462	827
2000	631	550	1137	1111	1089	1707
Total	1229	1066	2264	2257	2069	3416
	20 changes [a]					
100	41	31	57			
300	92	76	156			
700	198	170	374			
1000	293	252	541			
2000	646	556	1145			
Total	1270	1085	2273			

holds true for P2 even when the number of changes increases, enabling the estimation of reoptimization time when the number of nodes increases. However, in the case of MST-F, the reoptimization times increase at a slower rate than the number of nodes.

The last three columns of Table 5 present results classified by the number of changes by node degree, reinforcing the superiority of MST-F. As node degree increases, P2 performances become better compared to MST-F due to the increased number of updates performed with suboptimal path-weight labels.

In the case of changes to non-tree edge weights, Table 6 indicates that the non-greedy codes are better than P2, and in particular, MST-P has a slight edge over MST-F. This can be explained by the fact that in handling changes to non-tree edge weights, the non-greedy algorithms perform only a limited number of cycle traces and updates, starting from an old optimal tree, whereas P2 solved the problem anew. Since MST-P performs slightly better than MST-F, it also appears that the quality of the path-weight labels are not as crucial in this case. Furthermore, it is clear from Table 6 that overall performance is a function of the number of nodes, but the number of changes does not affect performance of the non-greedy codes to the extent that it does in the case of changes to in-tree edge weights.

8. Conclusions

Several variants of the non-greedy algorithm for the minimum spanning tree problem were developed and implemented using a combination of well known and novel labeling procedures. AA-P, an implementation of the ING algorithm using an artificial start and a partial update of the path-weight labels was identified to be the best non-greedy code. AA-P was compared with efficient implementation of the greedy algorithms and found to be dominated by all greedy approaches for all topologies and node degrees.

Algorithms and computational testing were also conducted for the reoptimization case. The non-greedy approaches proved quite successful in reoptimization and dominated greedy approaches on all topologies and node degrees. While the code MST-F was very efficient for modifications to edge weights of in-tree edges, the code MST-P was dominant for modifications to edge weights of non-tree edges.

Acknowledgements

We would like to thank Professor Doug Shier at the College of William and Mary for providing us with the random problem generator and the greedy codes P1, P2, and K.

This research was partially funded by the Center for Business Decision Analysis, the Hugh Roy Cullen Centennial Chair in Business Administration, the Office of Naval Research under contract N00014-87-K-0190, and the University of Minnesota Graduate School Grant-in-Aid under research budget 0959-3306-06.

References

Aho, A V , Hopcraft, J.E. and Ullman, J D (1974), *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA.

Barr, R , Glover, F., and Klingman, D (1979) "Enhancements of spanning tree labelling procedures for network optimization", *INFOR* 17/4, 16–33

Berge, C., and Ghouila-Houri, A. (1965) *Programming, Games and Transportation Networks*, Wiley, New York

Boruvka, O. (1926), "O jistem problemu minimalnim", *Práca Moravské Přírodovědecké Společnosti v Brne* 3, 37–58

Brennan, J J. (1982), "Minimal spanning trees and partial sorting", *Operations Research Letters* 1/3, 113–116.

Cheriton, D., and Tarjan, R.E (1976), "Finding minimum spanning trees", *SIAM Journal on Computing* 5/4, 724–742

Choquet, G (1938), "Etude de certains reseaux de routes", *C R Acad Sci Paris* 206, 310–313

Dijkstra, E W (1959), "A note on two problems in connection with graphs", *Numerische Mathematik* 1, 269–271

Edmonds, J (1971), "Matroids and the greedy algorithm", *Mathematical Programming* 1, 127–271

Gabow, H N , and Tarjan, R.E (1985), "Efficient algorithms for a family of matroid intersection problems", Technical Report CU-CS-214-82, University of Colorado

Gilsinn, J., and Witzgall, C (1973), "A performance comparison of labeling algorithms for calculating shortest path trees", NBS Technical Note 772, US Department of Commerce

Glover, F , Karney, D , and Klingman, D (1972), "The augmented predecessor index method for locating steppingstone paths and assigning dual prices in distribution problems", *Transportation Science* 6/2, 171–179

Glover, F, and Klingman, D (1974), "Finding minimum spanning trees with a fixed number of links at a node", in A Prekopa (ed), *Colloquia Mathematica Societatis*, North-Holland, New York, 425–439

Glover, F, Klingman, D, Phillips, N , and Schneider, R (1985), "New polynomial shortest path algorithms and their computational attributes", *Management Science* 31/9, 1106–1128

Glover, F , and Novick, B (1986), "The 2-quasi-greedy algorithm for cardinality constrained matroid bases", *Discrete Applied Mathematics* 13, 277–286

Golden, B L , Magnanti, T L , and Nguyen, H G (1977), "Implementing vehicle routing algorithms", *Networks* 7, 113–148

Gomory, R E , and Hu, T C (1961), "Multiterminal network flows", *SIAM Journal of Applied Mathematics* 9, 551–571

Gower, J C , and Ross, G J S (1969), "Minimum spanning trees and single linkage cluster analysis", *Applied Statistics* 18, 54–64

Graham, R L., and Hell, P (1985), "On the history of the minimum spanning tree problem", *Annals of the History of Computing* 7/1, 43–58

Haymond, R E , Jarvis, J P., and Shier, D R (1984), "ALGORITHM 613 Minimum spanning tree for moderate integer weights", *ACM Transactions on Mathematical Software* 10/1, 108–110

Haymond, R E , Jarvis, J P , and Shier, D R (1984) "Computational methods for minimum spanning tree algorithms", *SIAM Journal on Scientific and Statistical Computing* 5/1, 157–174

Held, M , and Karp, R.M (1970), "The traveling salesman problem and minimum spanning trees", *Operations Research* 18, 1138–1162

Held, M , and Karp, R M (1971) "The traveling salesman problem and minimum spanning trees II", *Mathematical Programming* 1, 6–25

Jarvis, J P , and Whited, D E (1983), "Computational experience with minimum spanning tree algorithms", *Operations Research Letters* 2/1, 36–41.

Johnson, D B (1975), "Priority queues with update and finding minimum spanning trees", *Information Processing Letters* 4/3, 53–57

Kershenbaum, A., and Sylke, R.V. (1972), "Computing minimum spanning trees efficiently", *Proceedings ACM National Conference*, 518–527

Kevin, V., and Whitney, M. (1972), "Algorithm 422: Minimal spanning tree [H]", *Communications of the ACM* 15, 273–274

Kruskal, J.B. (1956), "On the shortest spanning subtree of a graph and the traveling salesman problem", *Proceedings of the American Mathematical Society* 7, 48–50

Loberman, H., and Weinberger, A. (1957), "Formal procedures for connecting terminals with a minimum total wire length", *Journal of ACM* 4, 428–437

Pynn, C., and Warren, J.H. (1972), "Improved algorithm for the construction of minimal spanning trees", *Electronics Letters* 8, 143–144.

Prim, R.C. (1957), "Shortest connection networks and some generalizations", *The Bell System Technical Journal* 36/6, 1389–1401.

Spira, P.M., and Pan, A. (1975), "On finding and updating spanning trees and shortest paths", *Siam Journal on Computing* 4/3, 375–380.

Tarjan, R.E. (1983), *Data Structures and Network Algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol 44, Society for Industrial and Applied Mathematics, Philadelphia

Van Slyke, R., and Frank, H (1972), "Network reliability analysis Pt. I", *Networks* 1, 279–290.

Yao, A.C. (1975), "An O(e log v) algorithm for finding minimum spanning trees", *Information Processing Letters* 4, 21–23