MINIMAL SPANNING TREES WITH DEGREE RESTRAINTS

by

ARCHIBALD McFARLANE

B.Sc., University of British Columbia, 1963

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in the Department

of

COMPUTER SCIENCE

We accept this thesis as conforming to the

required standard

THE UNIVERSITY OF BRITISH COLUMBIA

October, 1968

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and Study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver 8, Canada

Date November 20, 1968

The purpose of this thesis is to develop a solution to the problem of determining the minimal spanning tree with degree restraints for a given non-directional graph.

Section 1 gives an introduction to the problem. A set of definitions describing the graphical terminology used in the body of the thesis, is presented along with a description of the problem. At the end of this section a few applications of the problem are given.

Section 2 outlines the method of solution used. The algorithm incorporates a branch and bound technique and this problem solving method is discussed in general in the first part of the section. Some other applications of branching and bounding are also discussed. Next, the complete algorithm is described along with a proof of optimality. A sample problem is worked through to illustrate the method of solution.

Two different minimal spanning tree algorithms, one by R.C. Prim, the other by J.B. Kruskal, are used in the main core of the solution algorithm. These two approaches are discussed with the aid of a sample problem, at the end of Section 2.

Computer programs were written to test the algorithms. Several sets of data were compiled for various sizes of graphs and values of degree restrictions. The results of these runs were tabulated and are discussed in Section 3. Next, a comparison is made of the method discussed here and a solution involving linear programming.

Section 3 also presents some useful heuristic approaches at sub-optimization which effectively reduce the amount of computation.

Section 4 summarizes the results of Section 3 and indicates the best approach to use for a specific problem.

I wish to thank Mr. H. Pachon of the Automatic Electric Company for his suggestion of the topic. I would also like to thank Dr. J.M. Kennedy for his assistance and encouragement during the writing of this thesis.

Finally I wish to thank the British Columbia Telephone Company for their financial aid and also the Department of Computer Science for making funds available during the summer of 1968.

1.1 Definitions

A graph G consists of a finite set P of nodes $P_1, P_2, \ldots, P_n$ and a set R of ordered pairs of distinct nodes of P. Each pair of nodes $(x,y) \in R$ is called link of the graph G.

Associated with each graph G, we have a distance matrix L with elements l_{ij}, $(i=1,2,\ldots,n;\ j=1,2\ldots,n;\ i \neq j)$ of the given graph. Node i is called the initial node of link l_{ij} and node j is called the terminal node. l_{ij} denotes the length of the link from i to j.

A path in a graph is a set of links $l_1, l_2, \ldots, l_s$ such that the terminal node of the l_i link corresponds to the initial node of the l_{i+1} link for $i=1,2,\ldots s-1$.

A graph is connected if there is a path joining every pair of nodes in the graph.

If $l_{ij} = l_{ji}$ for all $i, j \in P$ we say the graph is symmetric.

A link is said to be incident with the nodes it joins.

If x is an isolated node of graph G then $(x,i) \notin R$ and $(i,x) \notin R$ for all $i \in P$.

A connected graph containing n distinct nodes and n-1 distinct links with no isolated nodes is called a spanning tree. The degree of any node of a graph is equal to the number of links incident with it.

When the degree of every node of a tree is less than or equal to two, the tree defines a Hamiltonian path.

A feasible solution to a constrained optimization problem is a solution which satisfies the constraints of the problem.

A minimum spanning tree has a total link length no greater than that of any other spanning tree.

The minimum n-node spanning tree problem with degree restraints r, may be defined as follows:

Given n distinct nodes and an associated symmetric distance matrix, find the minimum spanning tree such that the degree of each node in the tree is less than or equal to r.

This type of problem occurs in the backplane wiring of computers. The connector pins are the nodes of the graph and the lengths of wire required to join any two pins are the links. The degree restraints arise from the physical limitations on the number of connections which can be made at any pin.

The problem also occurs in other electrical design applications, for example, in the layout of integrated circuitry and card modules.

When the degree restriction is equal to two, the problem becomes the determination of the shortest Hamiltonian path in the graph. This is closely related to the Travelling Salesman problem which is one of the 'unsolved' problems of combinatorial mathematics.

The number of distinct spanning trees in a complete symmetric graph with n nodes is n^{n-2}. The number of distinct Hamiltonian paths in this graph is $\frac{n!}{2}$. The probability of a minimum spanning tree being a Hamiltonian path is therefore given by the ratio of $\frac{n!}{2}$ to n^{n-2}. This ratio goes very quickly to its limit of zero. For example, when n=10, the probability is less than 0.04 and when n=15 it becomes less than 0.007. It is for this reason that the solution of the problem becomes very time consuming when the degree restrictions equal two and $n>15$.

The technique used to solve this restrained optimization problem is known as branch and bound. It is a means of progressing towards an optimal solution, with a significant reduction in the size of the set of feasible solutions requiring investigation.

A large set of feasible solutions normally exists when an objective function z is to be minimized, subject to a set of restraints. Each of these feasible solutions has a distinct z value. The branch and bound process breaks up this set of feasible solutions into smaller subsets and calculates a lower bound for the z values within each subset. These subset bounds are obtained by solving a simpler problem than the given restrained one. One of these subsets is selected and again partitioned into bounded subsets as above. The partitioning continues until a feasible solution, z' to the original problem is isolated. Subsets with a bound greater than or equal to z' are not investigated further. Subsets with a lower bound than z' are processed in the hope of discovering a smaller feasible solution. The optimal solution z* is reached when the bound on all subsets is greater than or equal to z*.

An important point in branching and bounding is the determination of the order in which the partitioned subsets are to be processed. This gives rise to two basic methods of subset selection. The first of these is the rather obvious one of choosing the subsets in the increasing order of their respective bound values. This would ensure a minimum amount of computation. The one disadvantage with this method is that a large amount of information must be retained at all times in order to determine and to define the subset which should be studied next.

The second method is that of investigating the subsets in some prearranged order until either a feasible solution is obtained, or the bound on a subset exceeds the length of a known feasible solution. In other words, the first subset of a partition is itself partitioned and so on until at, say, the kth level of partitioning a conclusion is reached as above. Now the second subset of this kth partition is investigated. This method is useful if the ordering of the subsets in each partition is such that there is a greater probability of finding the optimal solution in the initial subsets.

This latter method is the one used here and more will be said on the ordering of the partitions.

Branch and bound methods have been used to handle a variety of problems. E.L. Lawler and D.E. Wood (4) give an excellent account of several applications. They include integer linear programming, nonlinear programming, the quadratic assignment problem and the travelling salesman problem. There is also a note on applications outside the realms of mathematical programming; e.g. pure combinatorics.

The branch and bound technique used on the travelling salesman problem has been compared with other popular 'solutions' in an article by Bellmore and Newhauser (8). Here the branch and bound methods of Eastman (10), Little, et al (3), and Shapiro (11) are compared with the dynamic programming of Held and Karp (12) and the 'λ - optimization' of Lin (13). Shapiro's work appears to be the superior branch and bound approach and is rated very highly for the solution of symmetric problems of up to 40 nodes.

The algorithm is a search technique in which one partitions the set of spanning trees into subsets and calculates lower bounds on the length of all trees in a subset. In this way, spanning trees which meet the degree requirements are discovered, and the smallest one will be the optimal solution to the restrained problem.

The initial bound is found by solving the unrestricted minimum tree problem defined by the given distance matrix L with elements l_{ij} ($i=1,2,\ldots,n$; $j=1,2,\ldots,n$; $i\neq j$). If the solution complies with the degree restrictions, the restrained problem is solved trivially. If it does not satisfy the restraints, i.e. if there exists at least one node i in the tree of degree x with $x>r$, one branches into p subproblems where p is the number of ways of selecting x-r distinct links from x links. The above node i of degree x will have x links incident with it. Let these links be $l_1,l_2,\ldots,l_x$, arranged in the reverse order they are chosen in the minimum tree algorithm.

The p subproblems are created by prohibiting p distinct sets of x-r links from being included in minimum tree solutions. For subproblem 1, let $l_1=\infty, l_2=\infty,\ldots, l_{x-r}=\infty$; for subproblem 2, let $l_1=\infty, l_2=\infty,\ldots,l_{x-r-1}=\infty, l_{x-r+1}=\infty;\ldots$; for subproblem p let $l_{r+1}=\infty, l_{r+2}=\infty,\ldots,l_x=\infty$. These sets of prohibited links are the lexicographical orderings of the p selections of x-r links from links $l_1,l_2,\ldots,l_x$.

Subset number 1 then consists of the set of all trees for which links $l_1,l_2,\ldots,l_{x-r}$ are prohibited, subset number 2 prohibits links $l_1,\ldots,l_{x-r-1}, l_{x-r+1}$, etc. In short, we are investigating all the possible ways of forcing the degree of node i to satisfy the

degree restraints r. It is convenient to refer to a node whose degree exceeds the restrictions as a 'trouble' node, and the associated links as 'trouble' links. The minimum tree solutions to these p subproblems become the bounds for their respective subsets. If the solution to subproblem number 1 is not a feasible solution to the restrained minimum tree problem, we branch again. This continues until a feasible solution is reached in a subset k.

Now the algorithm examines the next subproblem in subset k as above. The subproblems are examined until either an improved feasible solution is found or the bound on a subset is larger than the length of the current best feasible solution, in which case the subset is rejected. When all possible subsets have been examined, the best feasible solution is the optimal solution to the minimum tree problem with degree restraints. In the next section an example is worked through.

It is convenient to use a pushdown stack to define the current subproblem. As soon as a branch occurs in the algorithm a new entry is placed on the top of the stack. This entry gives the ordered list of links connected to the trouble node which caused the branch. It also indicates which of these links are prohibited and which are allowed for the particular subproblem. This is accomplished by means of an x-r digit number, (x = degree of the trouble node) which indicates the current combination of prohibited links. This is best illustrated by an example.

With x = 6 and r = 2 we have C(6,4) = 15 subproblems to investigate. Initially we have the combination pointer set at 1234 which indicates that the first, second, third and fourth links of the ordered list in the stack entry are prohibited in the first subproblem.

The fifth and sixth links are allowed to enter the solution.

If this first subproblem yields an improved feasible solution, or exceeds the current bound, the combination pointer is changed from 1234 to 1235 giving a new set of allowed links and prohibited links for the second subproblem. If not, a new entry is added to the stack, depicting the next branching subset. Eventually, an entry on the top of the stack will exhaust all possible combinations. When this is the case, this entry is deleted and the next lower level becomes the new top of the stack. As before the next combination is generated, etc. In this manner the other thirteen subproblems for the above example will be duly investiaged; i.e. the pointer will take on the values 1236, 1245, 1246, ..., 2346, 2356, 2456, 3456. The algorithm terminates when the stack is empty.

The algorithm completes an iteration when either an improved feasible solution is found or a solution exceeds the current bound. If the same trouble node occurs more than once in the branching processes during an iteration, all the trouble links for that node are 'bunched' together at one branch. This cuts down unnecessary duplication of subsets.

If more than one node has a degree greater than r, the first one encountered by the unrestrained minimum tree algorithm is the one which determines the next set of subproblems.

The order in which the links are chosen in the minimum tree algorithm is extremely important. It affects not only the determination of the first trouble node but also the ordering of the trouble links for the combination generator.

This ordering of the links is dependent on the minimum spanning tree algorithm used and for this reason, two different methods are compared.

These two methods, one by R.C. Prim (7), the other by J.B. Kruskal (6), will be discussed in section 2.5.

The partitioning used explores all the necessary subsets, and ensures us of covering the entire set of feasible solutions. Clearly, this is due to the investigation of the complete set of possible combinations at each branching stage.

The algorithm is terminated when the bounds on all the subset problems are greater than or equal to the length of the best restrained minimal tree. To prove that this is the optimal restrained solution, it is only necessary to show that each time a branch is made, the subproblems defined will have optimal solutions greater than or equal to the lower bound at the branch.

Let us assume that a particular subproblem k, prohibiting x links yields a minimum spanning tree of length t. The distance matrix for this subproblem may be designated D_k.

Let this tree of length t be comprised of the links l_1, $l_2,\ldots,l_{i-1},l_i,l_{i+1},\ldots,l_s$, arranged in the order they are chosen by the minimum spanning tree algorithm. (Assume that the algorithm employs the rule of choosing the next largest link which does not form a closed loop with previously selected links.)

If we exclude any one of the links of this tree from our matrix D_k, giving a matrix D_k', and solve the new minimum tree problem, we will obtain a solution of length t' with $t' \geq t$.

To illustrate this, let us exclude the link l_i from D_k, and solve the new minimum tree problem defined on D_k'.

Links $l_1,l_2,\ldots,l_{i-1}$ will be chosen as before, but because l_i is prohibited, we must investigate the next largest link from our matrix D_k' to continue the algorithm. (The next 'largest' link l_m will have the relation $l_m \geq l_i$). The remaining sequence of minimum

spanning tree links will be the links $l_{i+i}, \ldots, l_j, l_i', l_{j+1}, \ldots, l_s$ with $l_{i+1}, \ldots, l_j, l_{j+1}, \ldots, l_s$ as above in the solution using the matrix D_k and l_i' a new link with $l_i' \geq l_i$. Thus the total length t' of this solution will be such that $t' \geq t$. Clearly this will also be the case if D_k' has more than one link of the D_k minimum tree prohibited.

Since every possible subset has a bound which is greater than or equal to the length z of the best restrained solution, no other restrained solution can exist with a length smaller than z.

Given the following 8x8 distance matrix (Fig. 1) and the degree restrictions r=2 we solve the unrestricted minimum tree problem. The minimum tree is given by the links 2-7, 4-8, 1-8, 5-8, 5-6, 3-8, 2-6 with total length 603. Clearly, node 8 has degree 4 and our solution is not a feasible one for the restricted problem.

Now the 4 links connected to node 8 are arranged in the order 8-3, 8-5, 8-1, 8-4 and the first of the C(4,2) = 6 subproblems is defined by setting link 8-3 $=\infty$ and link 8-5 $=\infty$. The distance matrix for this subproblem is shown in Fig. 2. The solution to this particular subproblem is defined by the links 2-7, 4-8, 1-8, 1-5, 5-6, 2-6, 3-4 with total length 781. This turns out to be our first feasible solution since the degree of each node is less than or equal to two. The current bound for our optimal solution is therefore 781. The second subproblem is defined by setting 8-3 $=\infty$, and 8-1 $=\infty$.

This search technique can best be illustrated by a tree diagram (see Fig. 3).

The nodes of the tree diagram represent unrestrained minimum tree subproblems and the branches tell which links have been set to infinity in the distance matrix; e.g. If we want to exclude the link x-y from a particular subproblem then $\overline{xy}$ would appear on a branch leading to it. The number at each node denotes the length of the particular solution. A square node indicates a feasible solution to the restrained problem.

Subproblems with minimum tree lengths greater than or equal to the current best feasible solution are not investigated further. Thus, the 2nd and 3rd subproblems with lengths greater than 781 are abandoned.

The 4th subproblem which has links 5-8 and 1-8 set to infinity in its distance matrix has a length of 735. The minimum tree for this subproblem is comprised of the links 2-7, 4-8, 1-5, 5-6, 3-8, 4-5 and 2-6. This solution is not a feasible one since node 5 has degree 3. Therefore this 4th subset is further partitioned into 3 smaller subsets; the 1st one excludes the link 4-5, the second excludes the link 6-5 and the last excludes the link 1-5. Now the 1st of these smaller subsets is examined. We continue in this fashion until all subsets (nodes of the tree diagram) have been investigated.

The optimum solution is comprised of the links 2-7, 4-8, 1-5, 3-8, 4-5, 1-6, 2-6 of length 767 and occurs when links 5-8, 1-8 and 6-5 are prohibited.

	1	2	3	4	5	6	7	8
1	-	576	374	357	63	174	511	37
2	576	-	854	205	474	186	5	739
3	374	854	-	332	872	587	846	156
4	357	205	332	-	167	671	597	16
5	63	474	872	167	-	142	326	61
6	174	186	587	671	142	-	983	865
7	511	5	846	597	326	983	-	622
8	37	739	156	16	61	865	622	-

FIGURE I 8 x 8 SAMPLE PROBLEM

	1	2	3	4	5	6	7	8
1	-	576	374	357	63	174	511	37
2	576	-	854	205	474	186	5	739
3	374	854	-	332	872	587	846	∞
4	357	205	332	-	167	671	597	16
5	63	474	872	167	-	142	326	∞
6	174	186	587	671	142	-	983	865
7	511	5	846	597	326	983	-	622
8	37	739	∞	16	∞	865	622	-

FIGURE 2 DISTANCE MATRIX FOR 1st SUB PROBLEM
LINKS (3,8) AND (5,8) PROHIBITED

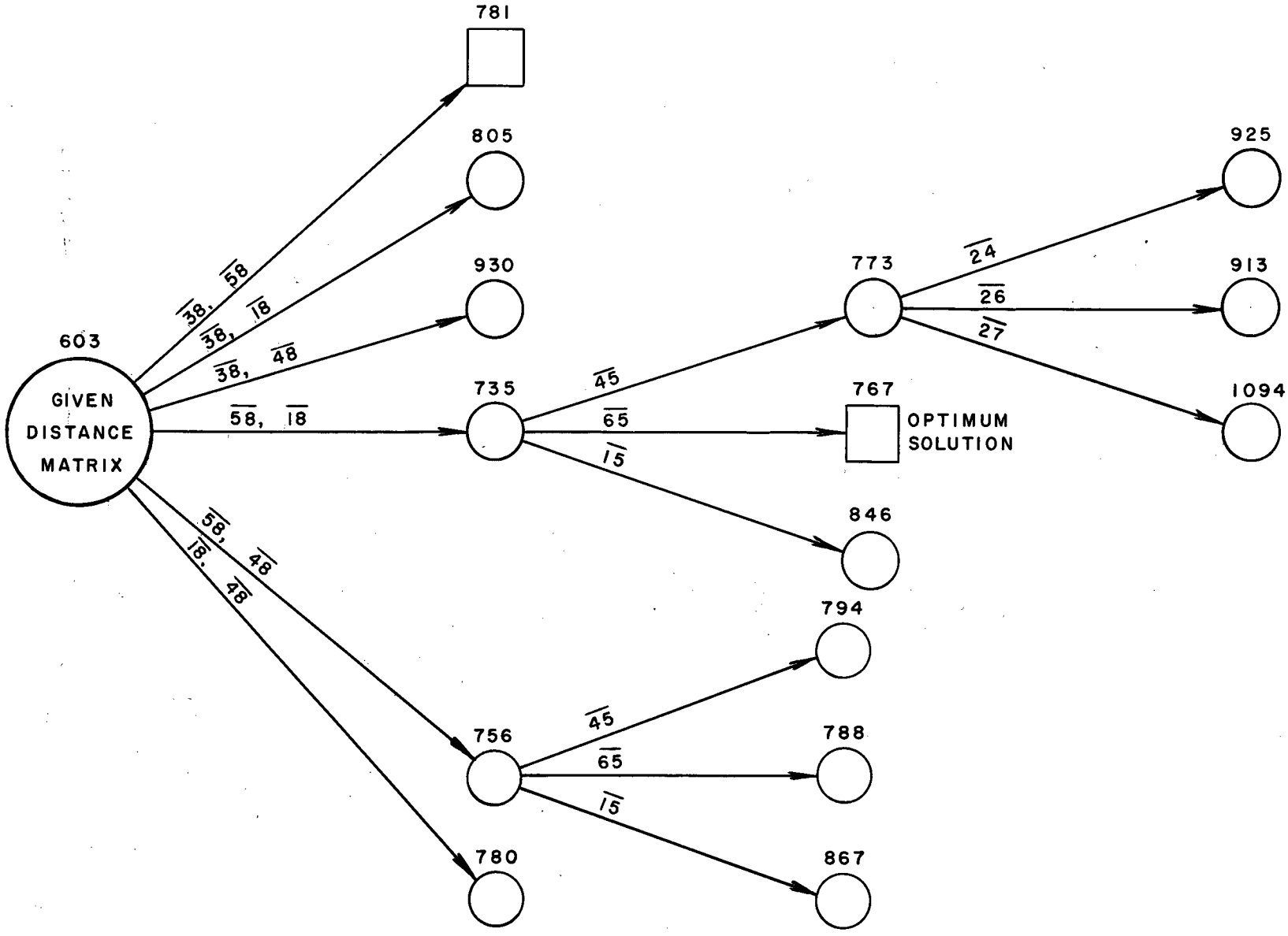

FIGURE 3 COMPLETE TREE DIAGRAM

The Branch and Bound method described solves a set of minimal spanning tree problems. Two different methods for generating this set are compared.

The first of these methods was developed by R.C. Prim (7). Basically, this algorithm starts with any given node and finds its nearest neighbour, thus forming the first link of the minimal tree. Then it finds the nearest node to this subtree by searching through the set of nodes not yet included in the subtree. This process continues until the full spanning tree is formed. At all stages we are dealing with a subtree, thus there is no need to check for closed loops or connectedness.

The algorithm is very fast and uses a minimal amount of core storage. A test program written in Fortran on an IBM 7044 found the minimum tree for an 80x80 complete distance matrix in less than one second. The big disadvantage with this algorithm is the fact that the minimal tree links are not chosen in increasing order of magnitude. Since this Branch and Bound method always examines the subset problems in order, there is a greater probability of finding the optimal solution earlier if the largest links connected to a trouble node are prohibited in the first problem of each subset. This is best accomplished if these links are given to the combination generator in order of increasing magnitude. Due to the nature of the combinations which are generated in lexicographical order, the first problem of a given subset will then exclude the x-r largest links (x being the degree of the trouble node for this subset with $x \geq r$), the second problem will exclude the x-r-1 largest links and the (x-r+1)th largest link and so on for the remaining problems.

The last problem of each subset will of course exclude the x-r smallest links.

The second algorithm developed by J.B. Kruskal (6) achieves this ordering. Initially all the links of the distance matrix are sorted in increasing order of magnitude. The first link in the sorted list becomes the initial link of the minimal spanning tree. Links which do not form closed loops with existing links are then chosen in order from the list until the complete tree is constructed.

The disadvantage here is that the sort time grows as the dimension n of the distance matrix increases. However, once the sort has been completed, the determination of the minimal tree is much faster than the Prim algorithm. The sort only needs to be done once in order to solve the many minimal tree subproblems which arise in the branching process. For a reasonable number of iterations the sort time is outweighed by the fast link selection and the overall Kruskal time becomes faster than the Prim time. The number of iterations necessary to complete the algorithm depends on both the dimension n of the distance matrix and the degree restriction r for the problem. Complete symmetric distance matrices of various dimensions with elements obtained from a random number generator were used to test both the Prim and the Kruskal methods of solution on an IBM 7044.. The results of these tests are given in Table 1.

A Library subroutine (14) is used to order the links in the Kruskal method. The sorting is accomplished by a merge-exchange technique and it is very fast. The number of iterations necessary for the Kruskal method to overcome its sort time and become faster than the Prim method, for the same number of iterations, was tabulated for each value of n and r. For example, with n = 40 and

r = 3, the Kruskal method would be faster than the Prim method if both took more than 25 iterations to terminate. An average value, over various sample sizes, was calculated for each n, r combination. In all cases fewer iterations were required on the average by the Kruskal method. Figures for three random matrices with $n \geq 50$ were shown for comparison. The Prim times to complete the algorithm for n = 20 and r = 2 were greater than 10 minutes on the average; therefore only 3 sets were given.

The Table indicates that the Kruskal method should be used for graphs with fifty or less nodes. Little can be said of problems with $n \geq 50$, since only a few examples were tested. In all cases, the Kruskal method terminated in a smaller number of iterations and each time, the number was well above the critical value for that class. In this range, the Kruskal sort times are becoming significant; for n = 80, the sort time was 11 seconds.

The following example with n = 8 and degree restrictions r = 2 given in Fig. 4 will illustrate the difference between the Kruskal and the Prim approach. The solution tree diagrams for both methods are given in Figs. 5 and 6.

The numbers within the nodes represent the specific iteration. The Kruskal approach finds the optimum solution in the first iteration and completes the algorithm in 8 iterations. The Prim method finds the optimum in the 10th iteration and terminates in 14 iterations. This is typical behaviour for Prim versus Kruskal as Tables 1 and 2 clearly indicate. Table 2 gives the average number of iterations necessary to reach an optimal solution for both the Prim method and the Kruskal method. The sample size

for each n, r set is included beneath these average figures. The ratio of the number of Kruskal iterations to the number of Prim iterations is also calculated.

In this case, the reason the Prim approach was slower to find the optimal solution was that the Prim algorithm failed to select the minimum tree links in increasing order of magnitude. For example, link 6-1 of length 172 units was selected before links 6-7 and 6-3 equal to 48 units and 83 units respectively. Thus the subproblems prohibiting 6-3 and 6-7, both smaller in length than link 6-1, were investigated first. This gave a bound of 970 units on the optimal solution (See Fig. 6). If the subproblem prohibiting link 6-1 had been investigated first, as in the Kruskal approach, the bound would have been 894 units and the $\overline{63}$ and $\overline{67}$ subsets would have been rejected with their bounds of 967 units and 944 units respectively. The same lack of ordering occurs at the node 7, (Prim ordering; 7-1, 7-6, 7-8, 7-4). Fortunately this does not create any extra iterations because the 970 bound is sufficient to reject the 9th and 10th subsets with bounds of 1,008 and 1,039 respectively.

		ITERATION TIME (SECS.)		AVERAGE SORT TIME(SECS.) KRUSKAL	CRITICAL NUMBER OF ITERATIONS TO OVERCOME K. SORT TIME	AVG. # ITERATIONS SAMPLE SIZE	
N	R	PRIM	KRUSKAL			PRIM	KRUSKAL
10	2	.014	.010	.07	18	173 29	147 29
15	2	.027	.018	.2	23	2412 19	1123 19
20	2	.046	.026	.4	20	12252 3	7672 3
20	3	.046	.026	.4	20	30 40	29 40
30	3	.094	.049	1.1	25	127 50	126 50
40	3	.167	.078	2.1	25	690 23	561 23
50	3	.260	.125	3.5	26	558 14	510 14
60	3	.360	.165	5.6	29	719 1	581 1
70	3	.500	.200	8.0	27	985 1	739 1
80	3	.620	.250	11.0	30	472 1	328 1

TABLE I COMPARISON OF PRIM AND KRUSKAL METHODS
NUMBER OF ITERATIONS AND ITERATION SPEED

N	R	AVERAGE NUMBER OF ITERATIONS FOR OPTIMUM SOLUTION		RATIO K/P
		KRUSKAL	PRIM	
10	2	82 29	114 29	0.745
15	2	769 19	1516 19	0.508
20	2	5422 3	6073 3	0.890
20	3	12 40	16 40	0.750
30	3	64 50	87 50	0.735
40	3	280 23	306 23	0.915
50	3	336 14	402 14	0.835

TABLE 2 COMPARISON OF ITERATIONS NECESSARY TO REACH OPTIMAL SOLUTION

	1	2	3	4	5	6	7	8
1	-	88	360	540	730	172	251	492
2	88	-	560	259	538	483	321	420
3	360	560	-	291	582	83	725	580
4	540	259	291	-	859	916	243	475
5	730	538	582	859	-	386	891	46
6	172	483	83	916	386	-	48	233
7	251	321	725	243	891	48	-	79
8	492	420	580	475	46	233	79	-

FIGURE 4 8 x 8 DISTANCE MATRIX
TO ILLUSTRATE DIFFERENCE
BETWEEN KRUSKAL AND PRIM

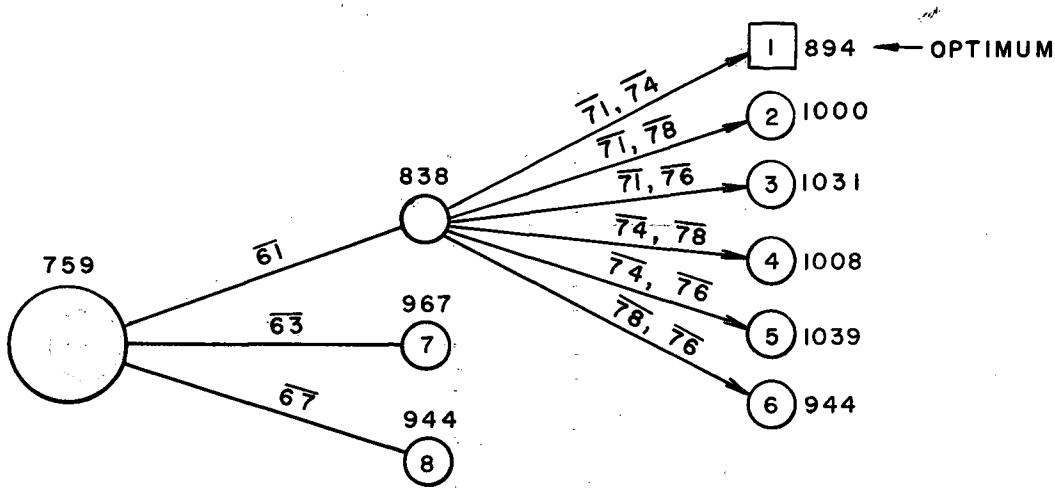

FIGURE 5 KRUSKAL TREE DIAGRAM

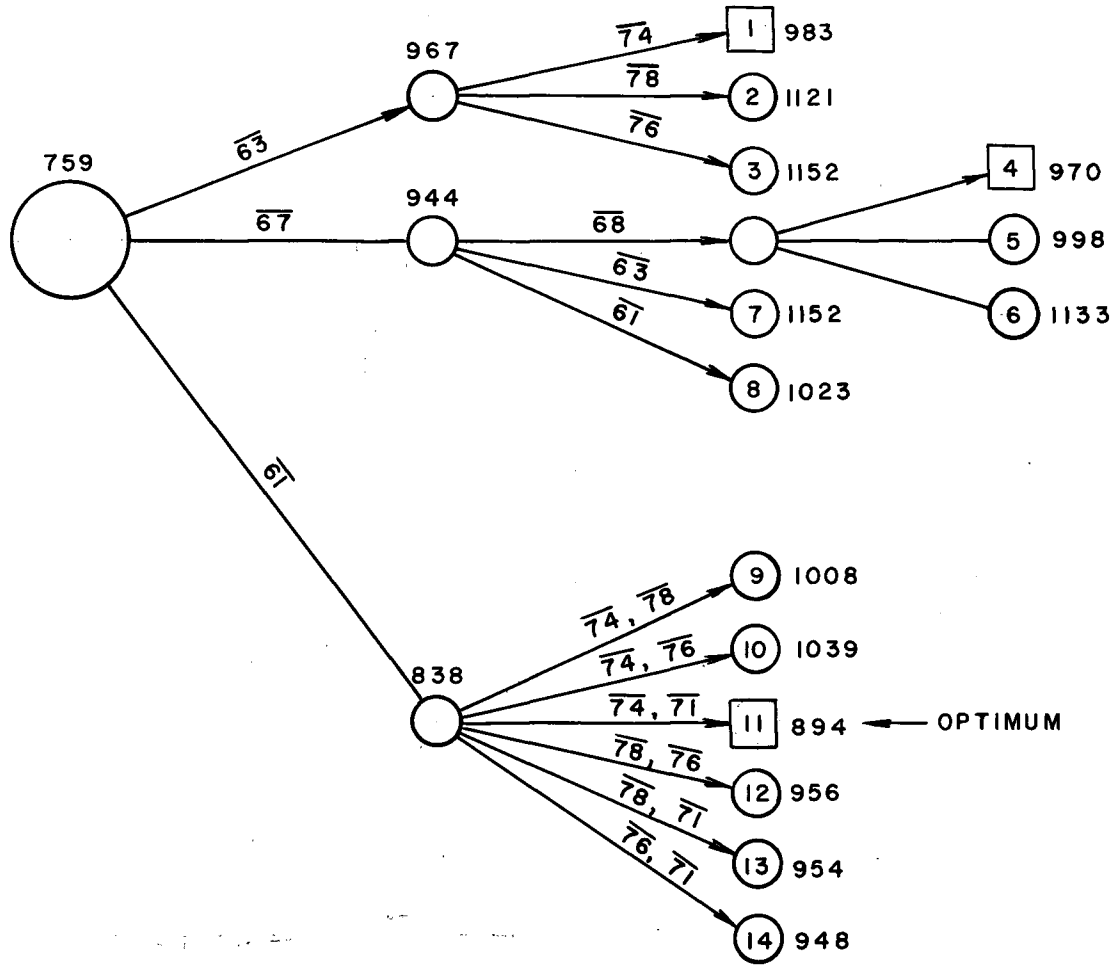

FIGURE 6 PRIM TREE DIAGRAM

Programs were written in Fortran for both the Prim approach and the Kruskal approach. The programs were tested on an IBM 7044 using complete symmetric distance matrices whose elements were random generated numbers. A description of the basic program is given in the appendix. Tables 3 through 10 give the results of several runs on various combinations of distance matrix sizes, n and degree restraints, r.

The tables give the number of iterations and the total times (in seconds) required both to terminate the algorithm and to find the optimum solution. These figures are given for both 'Kruskal' and 'Prim' as indicated. In almost all cases the results for each sample set give a wide range of values. However, for the larger samples, averages are calculated for both methods and included in the tables for comparison.

Table 3 with n = 10 and r = 2 illustrates the ability of the Kruskal method to find the optimal solution early. In 12 of the 29 cases, the optimal solution is found in 10 or fewer iterations and 5 of these 12 solutions are found in the first iteration. The average figures show the Kruskal method to be the superior one for this combination of n and r. The Prim method terminated faster than the Kruskal method on only one occasion and this occurred when the number of iterations for both methods was smaller than the critical value of 18 (See Table 1). For the same reason, the Prim method found the optimum solution faster in 5 cases. The average number of iterations necessary to terminate the Kruskal method was 147. This was felt to be a high figure since 22 of the 29 values were smaller than it. The same can be said of the Prim average of 173.

Table 4, clearly shows the superiority of the Kruskal method

for this particular n, r combination. The gap between the completion times of both approaches widens as the problems become more complex (i.e. when more branching steps are required to isolate a feasible solution). For example, the problem which required 4048 Kruskal iterations to terminate, required 7719 Prim iterations. Here the Kruskal method finished 137.1 seconds faster than the Prim method.

Occasionally, the Prim method will take fewer iterations than the Kruskal method ('Kruskal' took 1452 iterations to find an optimal solution which 'Prim' found in the first iteration). This is due to the fact that, in a few cases, the best restrained tree is not the one which deletes its largest superfluous links first at some branching stage. However, the average number of iterations over the sample set bears out in favour of the Kruskal choice of links. Furthermore, in 17 of the 19 instances, the Kruskal method terminates in fewer iterations. The average completion time for the Kruskal method was more than three times faster than that of the Prim method.

Table 5 gives a few results for n = 20 and r = 2. More success was again experienced with the Kruskal approach. Unfortunately, the large running times involved for both methods restricted the size of this data set. The four extra Kruskal solutions indicate problems of such a degree of complexity that the Prim algorithm failed to yield an optimal solution in a set time of 15 minutes. Once again we have a case where the Prim algorithm results in fewer iterations (first problem). The fact that the completion times are only 2 seconds apart, when the Kruskal method performs approximately twice as many iterations, illustrates the faster iteration time of the Kruskal algorithm (See Table 1).

Table 6 illustrates an 'easier' set of problems with n = 20

and $r = 3$. As a matter of fact, 6 of these 40 problems have a trivial solution; i.e. the solution to the unrestrained minimal spanning tree problem meets the degree restraints of two. For this class of problem, the effect of the link ordering in the Kruskal algorithm is not very significant. There are relatively few nodes of degree greater than three, and therefore most subsets will have three subproblems. The ordering of the links is more important when the subsets have more members, since it will take more iterations to reach a lower bound which might have been discovered in one of the first few subproblems, if the links had been ordered.

The results are more closely grouped and the averages give a better indication of the group. The average figures for the iterations are quite similar. In both cases, the average number of iterations necessary to reach the optimal value falls below the critical value of 20 for this group (See Table 1). For this reason, the optimum times for the Prim method are often the faster ones. (In 20 of the 40 cases the Prim method finds the optimal solution faster.)

In the set of Table 7, the Kruskal method has better average times than the Prim method. This is largely due to the faster iteration times of the Kruskal method, since the average number of iterations for both methods are quite similar. The two approaches have good success at finding the optimal solution in the first iteration. The Kruskal method accomplishes this 14 times and the Prim method, 10 times. There are four or five 'harder' problems in this set and a comparison of their solutions by both methods emphasizes the superiority of the Kruskal approach. All of the averages appear to be too high. For example, 43 of the 50 values are smaller than the average values for the number of

iterations necessary to terminate both the Prim and the Kruskal methods.

In Tables 8 and 9, the class of problems becomes more difficult. A comparison of the average completion times and the average optimum times will indicate the large savings in time, realized by using the Kruskal method over the Prim method. In every case in Table 8, the Kruskal method terminates faster than the Prim method. In Table 9, only one problem is solved faster by the Prim method and this is due to the fact that the total number of iterations necessary to terminate the algorithm (16 iterations) is less than the critical value (30 iterations).

Table 10 gives some results for a few matrices larger than 50 x 50. Clearly the Kruskal method should be used in this region at all times. The iteration times in Table 1 indicate that for this class of problems, the Kruskal approach can take more than twice as many iterations and still finish faster than the Prim approach.

The algorithms were not tested for problems with $n > 80$ and $r = 3$. In view of the previous results, it is felt that this region would best be dealt with using the Kruskal method. If the run times become too large for this class of problems, one of the Heuristic methods described in section 3.3 may be used.

The random numbers which formed the distance matrices for the preceeding examples were distributed in the range 0 to 1000.

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
33	6	35	6	0.42	0.15	0.52	0.13
12	2	12	2	0.22	0.10	0.20	0.05
50	2	57	15	0.57	0.10	0.85	0.25
88	33	114	59	0.92	0.42	1.62	0.90
37	24	55	47	0.42	0.30	0.82	0.70
124	52	180	106	1.24	0.59	2.68	1.65
34	1	36	9	0.37	0.07	0.53	0.17
542	464	566	480	6.05	5.29	7.75	6.60
24	12	55	46	0.30	0.20	0.78	0.67
77	28	80	36	0.84	0.35	1.15	0.53
25	20	23	18	0.30	0.25	0.35	0.28
22	17	20	7	0.29	0.22	0.30	0.10
21	1	147	127	0.25	0.09	2.00	1.75
933	166	913	237	9.42	1.82	12.40	3.28
22	1	26	1	0.27	0.07	0.37	0.02
84	29	79	10	0.87	0.37	1.07	0.15
68	61	73	63	0.74	0.67	1.05	0.92
28	6	31	15	0.37	0.17	0.48	0.27
20	1	32	15	0.25	0.07	0.47	0.25
14	1	99	93	0.20	0.08	1.42	1.33
185	65	211	118	1.90	0.77	3.08	1.75
160	139	338	316	1.65	1.45	4.70	4.40
270	8	302	87	2.69	0.17	4.12	1.25
951	928	956	934	10.05	9.82	13.27	12.97
8	1	35	30	0.14	0.07	0.53	0.47
77	35	135	71	0.82	0.44	1.92	1.03
207	128	227	204	2.22	1.49	3.25	2.93
146	138	146	138	1.65	1.57	2.04	1.93
14	10	33	26	0.22	0.17	0.50	0.40
			AVERAGES				
147	82	173	114	1.57	0.94	2.42	1.63

TABLE 3 RESULTS FOR N = 10 R = 2

SAMPLE SIZE = 29

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
2503	2083	3292	1370	41.7	35.0	88.9	37.3
880	716	1062	498	15.8	13.1	29.2	13.8
84	1	546	482	1.5	0.2	15.3	13.6
476	414	404	323	8.0	7.0	11.7	9.3
60	19	83	19	1.1	0.5	2.5	0.6
574	437	1379	1276	9.6	7.4	39.7	36.7
95	1	109	9	1.6	0.2	3.2	0.4
1942	1598	4758	4588	31.4	25.9	138.5	133.5
925	394	985	192	15.2	6.8	28.2	5.6
172	1	415	334	2.9	0.2	11.5	9.3
1344	1162	1832	428	23.3	18.8	52.8	12.4
4048	3606	7719	3821	79.8	70.8	216.9	109.2
114	1	115	1	2.0	0.2	3.4	0.1
37	28	40	31	0.7	0.6	1.2	0.9
623	370	1011	234	10.5	6.4	28.6	6.8
2871	1879	10155	8944	52.9	36.5	298.3	262.6
491	194	6366	5842	8.3	3.3	184.1	169.1
2091	1452	645	1	36.9	26.1	19.0	0.1
2004	256	4905	418	34.9	4.8	134.4	11.7
			AVERAGES				
1123	769	2412	1516	19.9	13.9	68.8	43.8

TABLE 4 RESULTS FOR N = 15 R = 2

SAMPLE SIZE = 19

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
2443	1762	1229	339	61.2	44.3	59.4	16.6
14565	9520	17572	14559	381.5	251.3	811.3	670.9
6007	4986	17956	3323	150.8	126.8	839.3	167.4
5984	3882			158.5	102.8		
6956	5737			187.3	156.2		
13698	11028			381.4	307.1		
29813	22697			781.4	594.7		
			AVERAGES				
13244	9935			300.3	226.2		

TABLE 5 RESULTS FOR N = 20 R = 2

SAMPLE SIZE = 7

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
65	15	65	23	1.9	0.7	2.8	1.1
7	2	7	2	0.5	0.4	0.4	0.1
10	1	10	1	0.6	0.3	0.5	0.1
46	3	46	3	1.5	0.4	2.0	0.2
44	26	47	30	1.3	0.8	2.2	1.4
16	14	22	20	0.5	0.5	1.1	1.0
31	20	34	26	1.0	.8	1.7	1.3
16	2	16	2	0.8	0.4	0.8	0.2
7	5	4	1	0.4	0.4	0.2	0.1
13	6	13	13	0.5	0.4	0.6	0.6
40	35	37	34	1.1	1.2	1.7	1.6
1	1	1	1	0.3	0.3	0.1	0.1
22	14	22	14	0.7	0.5	1.0	0.6
1	1	1	1	0.4	0.4	0.1	0.1
7	1	7	4	0.5	0.4	0.4	0.2
221	69	200	51	5.8	2.1	8.8	2.4
1	1	1	1	0.4	0.4	0.1	0.1
1	1	1	1	0.4	0.4	0.1	0.1
1	1	1	1	0.4	0.4	0.1	0.1
7	2	7	4	0.5	0.4	0.4	0.2
61	27	55	21	1.8	1.0	2.6	1.1
4	1	4	1	0.4	0.4	0.2	0.1
19	11	19	14	0.7	0.5	1.0	0.7
1	1	1	1	0.3	0.3	0.1	0.1
56	35	56	35	1.4	1.0	2.6	1.7
4	2	4	2	0.4	0.4	0.2	0.1
16	14	16	16	0.7	0.6	0.8	0.8
4	1	4	1	0.4	0.4	0.2	0.1
4	3	7	7	0.4	0.4	0.4	0.4
38	11	32	10	1.1	0.5	1.5	0.6
31	20	34	21	1.0	0.7	1.6	1.0

TABLE 6 RESULTS FOR N = 20 R = 3

SAMPLE SIZE = 40

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
19	14	46	44	0.7	0.6	2.2	2.1
10	5	10	7	0.5	0.5	0.6	0.4
137	38	122	73	3.5	1.3	5.6	3.4
22	2	22	2	0.9	0.4	1.1	0.1
79	71	79	73	2.3	2.1	3.7	3.4
77	29	98	56	2.0	1.0	4.7	2.8
7	3	16	16	0.5	0.4	0.8	0.8
4	1	4	1	0.4	0.4	0.2	0.1
10	2	10	4	0.5	0.4	0.6	0.3
			AVERAGES				
29	12	30	16	1.0	0.6	1.4	0.8

TABLE 6 (CONT'D) RESULTS FOR N = 20 R = 3

SAMPLE SIZE = 40

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
16	6	16	6	1.6	1.2	1.6	.7
690	476	842	764	35.2	24.8	82.5	74.6
100	33	100	200	6.3	3.1	10.0	2.2
65	1	65	10	3.9	1.1	6.1	1.0
49	30	49	30	3.2	2.4	4.8	3.0
112	51	100	47	6.0	3.3	9.4	4.4
52	1	109	7	3.4	1.1	11.4	.8
1	1	1	1	1.1	1.1	.1	.1
37	11	37	11	3.0	1.8	3.4	1.1
264	185	367	325	13.9	10.0	35.0	31.0
71	49	86	30	4.0	3.1	8.5	3.1
16	8	25	19	2.1	1.8	2.5	1.9
1	1	1	1	1.0	1.0	.1	.1
47	10	25	8	3.7	1.9	2.5	.9
650	513	620	527	30.2	24.0	60.6	51.6
28	17	121	119	2.5	2.0	12.3	12.1
31	11	28	16	2.8	1.9	2.9	1.7
13	2	13	2	1.5	1.0	1.2	.2
73	21	121	84	4.6	2.1	11.9	8.3
19	1	19	7	2.3	1.4	2.0	.8
13	1	13	1	1.6	1.1	1.2	.1
442	246	386	213	21.8	12.7	36.6	20.1
55	15	61	50	3.7	1.7	5.9	4.8
37	22	37	22	2.9	2.2	3.4	2.1
4	1	4	1	1.3	1.1	.4	.1
7	5	7	5	1.3	1.1	.7	.5
1	1	1	1	1.0	1.0	.1	.1
79	38	85	45	5.2	3.1	8.6	4.5
7	6	7	6	1.2	1.2	.7	.6
95	41	95	56	5.9	3.4	9.3	5.6
13	3	13	3	1.8	1.3	1.3	.4

TABLE 7 RESULTS FOR N = 30 R = 3

SAMPLE SIZE = 50

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
10	1	10	1	1.8	1.3	1.0	.1
4	1	4	1	1.4	1.3	.4	.1
41	6	41	6	2.6	1.1	3.9	.7
53	38	38	20	3.7	2.0	3.7	2.1
2070	949	1880	1080	95.4	44.5	177.6	102.3
274	254	304	304	15.1	14.1	30.3	30.2
10	1	10	4	1.6	1.2	1.1	.5
13	9	13	9	1.8	1.6	1.3	.9
65	59	28	20	4.2	3.9	2.9	2.1
7	1	7	1	1.5	1.3	.7	.1
430	27	303	237	22.5	3.1	29.1	22.7
4	2	4	2	1.3	1.2	.4	.2
104	8	113	10	6.2	1.8	10.9	1.1
16	3	16	3	1.6	0.9	1.7	.3
13	5	13	1	1.9	1.5	1.5	.3
7	1	7	1	1.8	1.7	.7	.2
7	1	7	4	1.6	1.3	.8	.5
34	21	37	17	3.0	2.4	3.8	1.8
31	5	40	16	2.8	1.6	4.0	1.7
			AVERAGES				
126	64	127	87	7.0	4.2	12.2	8.1

TABLE 7 RESULTS FOR N = 30 R = 3

SAMPLE SIZE = 50

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
127	98	199	169	11.7	9.6	33.8	28.8
1418	1299	1192	724	113.1	104.3	200.0	121.9
331	131	319	116	29.9	14.5	55.6	20.1
5529	2171	6398	1196	405.9	165.5	1086.4	201.8
34	14	38	1	3.8	2.4	6.8	.6
25	1	64	1	3.6	2.1	10.9	.3
441	317	418	294	37.4	27.7	70.2	49.5
603	396	1186	1051	46.4	31.5	192.6	170.3
85	76	85	77	8.1	7.5	14.6	13.3
316	56	352	148	24.5	6.0	58.4	24.5
61	46	73	56	6.5	5.4	12.9	9.9
214	17	238	20	17.9	2.9	38.3	3.4
982	5	1048	92	72.9	2.4	176.0	15.9
201	69	297	139	16.2	6.8	48.4	22.8
399	200	522	272	30.6	16.1	86.6	45.6
25	1	25	10	4.0	2.3	4.1	1.7
49	25	58	38	5.7	4.0	10.1	6.7
1237	1082	2604	2257	101.0	88.8	428.9	372.2
34	11	58	37	4.2	2.6	10.3	7.0
130	125	67	49	12.6	12.2	10.9	8.0
106	11	112	11	10.7	3.2	19.9	2.1
550	284	493	269	44.3	23.5	82.3	45.5
16	5	22	13	3.3	2.5	3.7	2.2
			AVERAGES				
561	280	690	306	44.1	23.6	115.7	51.0

TABLE 8 RESULTS FOR N = 40 R = 3

SAMPLE SIZE = 23

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
94	34	100	57	14.1	7.0	26.4	15.2
830	333	926	643	95.2	40.6	238.6	166.5
40	2	52	2	8.7	4.5	13.3	.8
163	116	88	59	22.9	17.6	22.9	15.5
28	9	64	57	6.4	4.4	17.2	15.5
145	79	382	345	21.9	14.5	98.5	89.1
22	1	37	34	5.0	2.8	9.7	8.9
3221	2284	3437	2629	365.5	261.4	894.1	685.0
63	18	63	18	11.4	6.5	15.9	4.7
826	536	864	535	97.2	64.2	219.7	135.7
241	224	283	266	31.2	29.3	70.6	66.4
16	1	16	4	4.8	3.2	4.5	1.4
427	380	490	467	51.8	46.5	130.3	124.4
1033	679	1009	509	122.0	81.2	282.5	143.0
			AVERAGES				
510	336	558	402	61.3	41.7	146.0	105.0

TABLE 9 RESULTS FOR N = 50 R = 3

SAMPLE SIZE = 14

ITERATIONS				RUNNING TIMES (SECS)			
KRUSKAL		PRIM		KRUSKAL		PRIM	
END	OPTIMUM	END	OPTIMUM	END	OPTIMUM	END	OPTIMUM
	N=60		R=3				
581	2	719	46	93.9	6.4	260.5	17.1
298	259	1597	1578	51.4	45.6	584.4	577.6
	N=70		R=3				
739	611	985	918	171.9	144.0	494.5	460.7
	N=80		R=3				
328	140	472	274	90.0	52.5	295.6	171.8
1009	873	487	224	269.2	234.4	302.0	138.7

TABLE 10 RESULTS FOR N > 50 R = 3

A different approach at solving the problem has been proposed by N. Deo and S.L. Hakimi (5). They set up the problem as a linear program in the following manner.

The total length of n-1 links chosen from the distance matrix must be minimized subject to the following conditions.

(1) Every vertex must have degree less than an equal to the problem restraint, r.

(2) There is no isolated vertex in the graph.

In order to define the linear program, it is necessary to introduce many additional variables. These variables are bounded and can only assume the values 0 or 1. They end up with a simplex tableau involving n(n+1) variables in $\frac{n(n+3)}{2} + 1$ equations, which for n = 15 becomes 240 variables in 136 equations.

The above restraints do not guarantee that the solution to the linear program will yield a tree. There is a possibility of closed loops existing in the solution.

Deo and Hakimi incorporate a tree test algorithm to determine the feasibility of the solution. If the solution fails this tree test, the next smallest vector is forced into the solution and the linear program is iterated again.

An example is given of a 14 x 14-incomplete-symmetric graph with a degree restraint of two. It is solved on an IBM 709 in a matter of 249 seconds. The same problem was solved by the Prim method in 47 seconds and by the Kruskal method in 18 seconds on an IBM 7044. The internal logic and arithmetic organization of the two machines is quite similar but the cycle time of the 7044 is six times faster than that of the 709. Therefore a rough estimate of

the running time of the Deo and Hakimi solution on the 7044 would be about 40 seconds.

One advantage of the branch and bound approach is that we will always obtain a reasonably good upper bound on our optimal solution if we run out of time. With this linear programming approach, we have no such estimate if the problem takes longer than a set time.

The following is an account of a few schemes designed to decrease significantly the amount of computation involved in a problem at the risk of failing to find the optimal solution, and accepting a near optimal solution in a few cases.

The first of these guarantees that our answer will be within a prespecified amount of the optimal solution. Suppose we are willing to accept a feasible solution which differs from the optimal solution by no more than 10%. If a feasible solution is discovered with a total length of 2000 units, then we can reject all subsets with bounds of 1819 or more (1.10 x 1819 = 2000.9>2000).

A few of these suboptimization runs were made, both with the Kruskal method and the Prim method. The results are given in Tables 11, 12 and 13. In each of these runs a 10% suboptimization limit is allowed. The actual deviation from the known optimal solution is tabulated. At the bottom of each table, the average amount of work and time saved is given for the data set.

Table 11 shows an average deviation of 2% from the optimums and in 11 of the 20 cases the optimal solution was found. This set of data contains more difficult problems than the set of Table 4, as indicated by the higher average figures. The suboptimization effectively reduces these averages by more than 60% in all cases.

Table 12 gives a few results for n = 20 and r = 2. The savings there are all in excess of 75%.

The 10% suboptimization run using the Prim method is presented in Table 13. In 16 of the 20 cases, the first feasible solution reached is within the 10% limit. This indicates that the

method gives a relatively good initial estimate of the optimal solution. The Kruskal method also gave this good first guess, but because of the 2.1 second sort time, the average completion time over the same set was greater than the Prim method.

The average amounts saved in the Prim case were all greater than 85% with the average deviation from the optimal less than 4%.

The sort algorithm used for the Kruskal algorithm has a speed which is proportional to N ln N, where N is the size of the unsorted list. Since we are dealing with n x n complete symmetric matrices, these lists are of size $\frac{n(n-1)}{2}$. Therefore the net effect has the sort times increasing by the order of n^2 ln n (see Table 1).

The Kruskal algorithm is faster for sparse matrices than for complete matrices since the sort time is reduced due to the smaller list of links requiring sorting. At the moment, the Prim method is not set up to deal with sparse matrices. The algorithm is matrix oriented, with the position of a link in the distance matrix representing the nodes which the link joins. Sparse matrices could be handled more efficiently if a list of links was formed as in the Kruskal algorithm.

The difference between the completion time and the time when the optimal solution is found is often quite large (see Tables 3 to 10). This suggests concluding the algorithm after some set time and taking our current best feasible solution as an estimate of the optimal solution. The cases for $n > 50$ and $r = 3$ appear to back up this approach, in that many of the initial feasible solutions are quite close to the optimal.

In the five problems with $n > 50$, four produced first estimates within 10% of their optimum solutions and one gave a first estimate within 15% of its optimum solution.

A time limit of 30 seconds was imposed on the Kruskal method for the 20 problems of Table 11. The optimal solution was found in 12 cases. The remaining problems averaged solutions within 13% of the optimum. The largest deviation was 35%. The total time for the entire set was 433 seconds. The time for the 10% suboptimization run was 506 seconds and the complete algorithm took 1392 seconds.

The same time limit was used on the 'easier' set of Table 4. The optimal solution was found in 16 of the 19 cases. The remaining three answers were within 0.5%, 13% and 11% respectively. The total time saved over the basic algorithm was 111 seconds.

A survey of the available data was made to see what per cent of the total number of links was actually processed in arriving at an optimal solution. This survey was taken on the Kruskal results and is summarized in Table 14. The average position in the sorted list of links, where the final link of an optimal solution occurred, was obtained for each set of data. These figures were expressed as a percentage of the total number of links available ($\frac{n(n-1)}{2}$). Percentage figures were also obtained for the maximum list positions of the optimum solutions in each set. From these results, a proposed limit, on the percentage of links requiring investigation, was estimated for each n, r combination. For example, from Table 14, for n = 50, r = 3, it is estimated that we only need to examine the smallest 245 (0.20 x 1225 = 245) links of the distance matrix.

The average number of links prohibited in an optimal solution also appeared to be an interesting statistic to investigate. Table 15 gives the average value and the maximum value of the number of links set to ∞ at the optimal solution for each n and r studied.

It was felt that a saving in time could be realized by imposing these two limiting conditions on the algorithm. Viz., limit the

set of links studied to the smallest x% and at the same time limit the number of links, y, set to infinity (x and y are taken from Tables 14 and 15 respectively).

A few runs were tried using both the maximum and the average values given in Table 15 for the y limit.

There was no effective reduction in the number of iterations and in some cases, extra iterations were required. It was felt that the limits imposed on the algorithm, excluded some subsets which would have given us a better estimate of the optimal solution at an earlier stage. Thus our bound value for rejecting subsets was higher and consequently some additional subsets were examined. It appeared that the number of subsets rejected by this limiting strategy was approximately equal to the number that would have been rejected if these lower bounds had been established earlier.

A combination of some of these ideas could be used where further reduction in time is desired. For example, the percentage suboptimization could also be used with a lower time limit since the complete solution tree is much smaller than it is in the main algorithm.

ITERATIONS				RUNNING TIMES (SECS.)				% Greater than Optimum
Kruskal		Kruskal Suboptimum		Kruskal		Kruskal Suboptimum		
End	Optimum	End	Optimum	End	Optimum	End	Optimum	
2091	1452	340	226	36.9	26.1	6.1	4.2	0
2004	256	910	1	35.0	4.8	16.0	.2	5
20596	14951	8586	2291	371.1	272.4	152.6	42.1	6
3276	2989	1103	509	55.4	50.8	18.5	9.0	5
1914	1578	705	443	38.2	31.4	14.1	8.8	5
313	1	66	1	5.3	.2	1.2	.2	0
8171	1100	1538	449	134.1	19.1	24.8	7.8	0
1779	1141	322	223	30.7	20.1	5.5	4.0	0
19720	7274	9349	4166	337.7	128.5	158.1	73.0	2
3992	2588	1370	908	72.4	48.6	24.8	17.2	2
4866	3380	2034	1376	84.3	59.7	34.8	24.1	4
554	160	222	36	9.2	2.8	3.7	0.8	3
62	1	15	1	1.2	.2	.4	.2	0
218	1	20	1	3.6	.2	.4	.2	0
588	555	275	266	10.6	10.1	4.9	4.8	0
1689	1018	526	393	28.5	17.5	9.0	6.8	0
753	636	367	329	13.8	11.9	6.5	6.0	0
3614	2766	720	589	63.3	49.3	12.9	10.9	0
192	1	16	1	3.5	.2	.4	.2	0
3358	2978	641	1	58.3	52.0	11.2	3.2	7
			AVERAGES					
3988	2241	1456	610	69.6	40.2	25.3	11.2	2.0
			AMOUNT SAVED					
		63%	73%			62%	72%	

TABLE 11 RESULTS FOR N = 15 R = 2

SAMPLE SIZE = 20

ITERATIONS				RUNNING TIMES (SECS.)				% Greater than Optimum
Kruskal		Kruskal Suboptimum		Kruskal		Kruskal Suboptimum		
End	Optimum	End	Optimum	End	Optimum	End	Optimum	
2443	1762	613	324	61.2	44.3	15.4	8.3	5
14565	9520	3504	64	381.5	251.3	89.3	2.4	6
5984	3882	1031	715	158.5	102.8	28.1	19.9	3
6007	4986	1553	543	150.9	126.8	37.7	14.3	1
			AVERAGES					
7249	5037	1675	412	188.0	131.3	42.6	11.2	3.7
			AMOUNT SAVED					
		77%	92%			77%	91%	

TABLE 12 RESULTS FOR N = 20 R = 2

SAMPLE SIZE = 4

ITERATIONS				RUNNING TIMES (SECS.)				% Greater than Optimum
Prim		Prim	Suboptimum	Prim		Prim	Suboptimum	
End	Optimum	End	Optimum	End	Optimum	End	Optimum	
199	169	22	1	33.8	28.8	3.7	.3	4
1192	724	179	158	200.0	121.9	30.1	26.7	1
319	116	19	11	55.6	20.1	3.2	1.9	0
6398	1196	467	1	1086.4	201.8	79.0	.5	4
38	1	29	1	6.8	.6	5.2	.7	0
64	1	13	1	10.9	.3	2.2	.3	0
418	294	25	1	70.2	49.5	4.3	.4	2
1186	1051	79	1	192.6	170.3	13.3	.4	6
85	77	31	17	14.6	13.3	5.3	3.0	4
352	148	112	1	58.4	24.5	18.7	.3	8
73	56	25	1	12.9	9.9	4.6	.3	9
238	20	25	1	38.3	3.4	4.1	.3	3
1048	92	397	1	176.0	15.9	66.4	.6	8
297	139	16	1	48.4	22.8	2.7	.3	2
522	272	78	1	86.6	45.6	13.1	.5	7
25	10	13	1	4.1	1.7	2.1	.2	1
58	38	13	1	10.1	6.7	2.3	.3	2
2604	2257	436	1	428.9	372.2	72.0	.3	9
58	37	19	1	10.3	7.0	3.3	.5	7
67	49	40	28	10.9	8.0	6.5	4.6	0
			AVERAGES					
762	337	102	12	127.8	56.2	17.1	2.1	3.9
			AMOUNT SAVED					
		86%	96%			86%	96%	

TABLE 13 RESULTS FOR N = 40 R = 3

SAMPLE SIZE = 20

N	R	% LINKS SEARCHED FOR OPTIMUM SOLUTION		PROPOSED LIMIT
		AVERAGE	MAXIMUM	
10	2	38%	58%	60%
15	2	32%	60%	60%
20	2	20%	23%	50%
20	3	17%	36%	40%
30	3	14%	21%	30%
40	3	10%	13%	20%
50	3	9%	15%	20%
60	3	–	12%	15%
70	3	–	10%	15%
80	3	–	6%	10%

TABLE 14 KRUSKAL SEARCH LIMITS

N	R	NUMBER OF LINKS SET TO ∞ AT OPTIMUM	
		AVERAGE	MAXIMUM
10	2	6	11
15	2	12	23
20	2	18	26
20	3	2	8
30	3	3	9
40	3	6	11
50	3	6	10
60	3	12	12
70	3	6	10
80	3	10	10

TABLE 15 NUMBER OF PROHIBITED LINKS FOR N,R.

A branch and bound algorithm has been developed to solve the problem of finding the restrained minimal spanning tree for a symmetric graph.

Two options in the basic algorithm are presented and both of these methods are used to solve a wide variety of problems. The results indicate that the Kruskal method is the more efficient one of the two. The relative efficiency of the Kruskal method over the Prim method is directly proportional to the degree of complexity of the problem. In many of the 'difficult' problems with degree restrictions equal to two and distance matrix dimensions of twenty, the Prim method failed to terminate within a fixed time of twenty minutes. The Kruskal method solved this same set in an average time of five minutes a problem.

On the other hand, the two methods are quite comparable for the 'easier' class of problems with degree restraints of three and matrix dimensions no greater than thirty.

In most cases, the Kruskal method gives a better initial estimate of the optimal solution than the Prim method.

When the degree of difficulty of a problem is such that even the Kruskal approach fails to yield an optimal solution in a fixed length of time, then one or more of the heuristic methods of section 3.3 may be used. Of these methods, the one which appears to give the largest time savings combined with smallest average deviation from the optimal solution, is the percentage suboptimization technique.

The Kruskal method is the superior one for dealing with sparse matrices, since the sort time is less than for complete matrices due to the reduction in the number of links.

The algorithms presented here were only developed for symmetric matrices, but the branch and bound portion would be exactly the same for the nonsymmetric case. Only the minimal spanning tree algorithms would have to be altered to deal with nonsymmetric matrices.

1. Edwin F. Bechenbach, "Applied Combinatorial Mathematics" (Wiley).

2. Claude Berge, "The Theory of Graphs and Its Applications", A. Doig (trans.), Methnen, London, 1962.

3. J.D.C. Little, K.G. Murtz, D.W. Sweeney, and C. Karel, "An Algorithm for the Travelling Salesman Problem", Operations Research 11, 972-989 (1963).

4. E.L. Lawler and D.E. Wood, "Branch-and-Bound Methods: A Survey", Operations Research 14, 699-719 (1966).

5. Narsingh Deo and S.L. Hakimi, "The Shortest Generalized Hamiltonian Tree", 3rd Allerton Conference on Circuit and System Theory (1965).

6. J.B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Travelling Salesman Problem", Proceedings of American Mathematical Society, Vol. 7, pp 48-50, 1956.

7. R.C. Prim, "Shortest Connection Networks and Some Generalization, Bell System Technical Journal, Vol. 36, pp 1389-1401.

8. M. Bellmore and G.L. Newhauser, "The Travelling Salesman Problem: A Survey", Operations Research 16, 538-558, (1968).

9. C.A.C.M. Algorithm No. 154.

10. W.L. Eastman, "Linear Programming with Pattern Constraints", Ph.D. Dissertation, Harvard, 1958.

11. Shapiro, D., "Algorithms for the Solution of the Optimal Cost Travelling Salesman Problem", Sc.D. Thesis, Washington University, St. Louis, 1966.

12. M. Held and R.M. Karp, "A Dynamic Programming Approach to Sequencing Problems", Journal S.I.A.M. 10, 196-210 (1962).

13. Lin, S., "Computer Solution of the Travelling Salesman Problem", Bell System Techn. G 44, 699-719 (1966).

14. University of British Columbia Computing Centre, M1 UBC SORT.

APPENDIX

Description of the Computer Program

The algorithm was programmed in Fortran IV for an IBM 7044 computer. Fig. 7 shows a general block diagram of the program.

Box 1 of the figure denotes the minimal spanning tree algorithm where the tree links are selected from the given distance matrix. Either the Kruskal or the Prim algorithm is used here.

Box 6 indicates the routine for 'bunching' the links from like trouble nodes. These sets of links are stored temporarily in a matrix until a feasible solution is reached, or the current bound is exceeded (box 2). Then they are added to the pushdown stack in consecutive levels, each level representing a different trouble node, (box 4).

The following routine is incorporated into the algorithm to save unnecessary redetermination of the minimum tree links which were chosen before the degree restraints were exceeded. As soon as a node exceeds the degree restrictions, the status of the current subproblem is recorded; e.g. partial tree length, selected links, trouble node, etc. as shown in box 3. Then, using switch 1, the minimal tree is completed without further checking of the restraints. If the length of this tree is smaller than the current feasible bound, the superfluous links at the trouble node are excluded from the distance matrix, (box 5). The minimum spanning tree algorithm is then restarted using the stored partial tree information, (box 7). The time savings become very significant when a trouble node occurs more than halfway through the minimum tree algorithm.

As indicated in box 8, the algorithm terminates when the pushdown stack is empty.

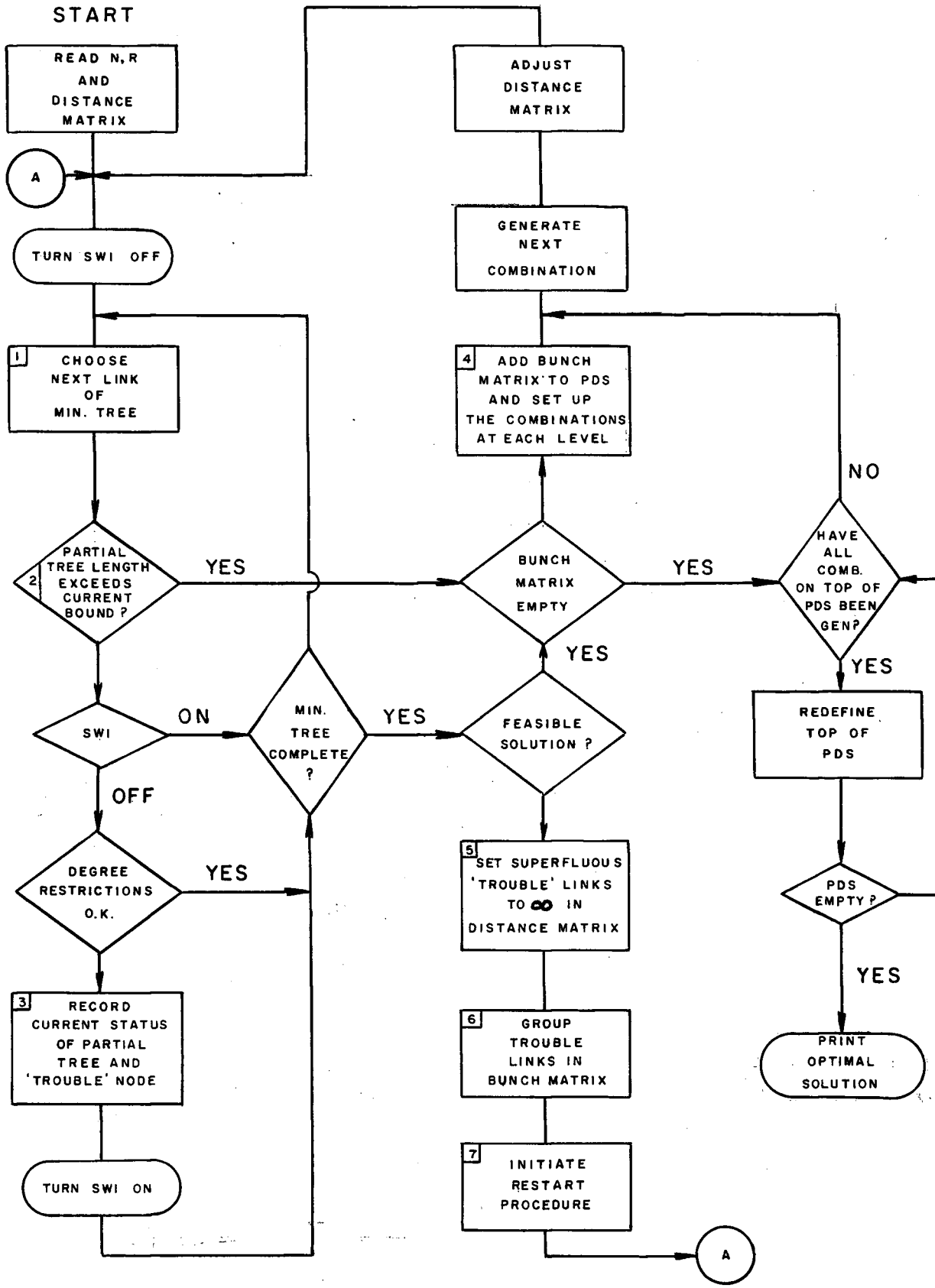

FIGURE 7 PROGRAM BLOCK DIAGRAM