AD-A258 301

IMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
August 1992

3. REPORT TYPE AND DATES COVERED
THESIS/~~DISSERTATION~~

1

4. TITLE AND SUBTITLE
An Algorithm for the Solution of a Traveling Salesman Problem to Minimize the Average Time to Demand Satisfaction

5. FUNDING NUMBERS

6. AUTHOR(S)
James Patrick Ryan, Captain

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
AFIT Student Attending: Texas A & M University

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/CI/CIA- 92-093

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFIT/CI
Wright-Patterson AFB OH 45433-6583

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

92-31145

64p8

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release IAW 190-1
Distributed Unlimited
ERNEST A. HAYGOOD, Captain, USAF
Executive Officer

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

DTIC
ELECTE
DEC 0 9 1992
S D
A

14. SUBJECT TERMS

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT

18. SECURITY CLASSIFICATION OF THIS PAGE

19. SECURITY CLASSIFICATION OF ABSTRACT

20. LIMITATION OF ABSTRACT

92

AN ALGORITHM FOR THE SOLUTION OF A TRAVELING SALESMAN PROBLEM TO MINIMIZE THE AVERAGE TIME TO DEMAND SATISFACTION

A Thesis

by

JAMES PATRICK RYAN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 1992

Major Subject: Industrial Engineering

AN ALGORITHM FOR THE SOLUTION OF A TRAVELING SALESMAN PROBLEM TO MINIMIZE THE AVERAGE TIME TO DEMAND SATISFACTION

A Thesis

by

JAMES PATRICK RYAN

Approved as to style and content by:

Alberto Garcia-Diaz (Chair of Committee)	Cesar O. Malave (Member)
Charles E. Gates (Member)	Gary L. Hogg (Head of Department)

August 1992

ABSTRACT

An Algorithm for the Solution
of a Traveling Salesman Problem
to Minimize the Average Time
to Demand Satisfaction. (August 1992)

James Patrick Ryan
B.S., U.S. Air Force Academy
Chair of Advisory Committee: Dr Alberto Garcia-Diaz

This paper presents a solution approach for solving a modified *traveling salesman problem*. In this problem, each city which the salesman must visit contains a quantity of demand (in units) which the salesman must satisfy. The objective of the salesman is to minimize the average time until satisfaction for each unit of demand. The solution approach presented is a branch-and-bound approach in which the total set of feasible tours are broken down into subsets. Lower bounds are the calculated for each subset and compared against a known upper bound to determine if this subset can be fathomed or if a improved upper bound has been found. The solution approach presented ensures optimality. A computer code was created and is presented which implements the solution approach developed.

ACKNOWLEDGEMENTS

I would like to thank the faculty of the Industrial Engineering Department for all their assistance throughout my research on this project. I would especially like to thank ny committee chairman, Dr. Alberto Garcia, for his time and the advice he gave me in completing this thesis. I am also grateful to my friends and office mates who lent me their ideas and critiques throughout this effort. Finally, my greatest appreciation goes to my wife, Christine, who endured the long hours I worked to complete this thesis and her unending support throughout.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

One of the most well known problems in Operations Research is the Traveling Salesman Problem (TSP). In the classical TSP, a 'salesman' starting at a home city is required to visit every city in a given group once and only once and then return to the home city. The objective of this problem is to find the optimal sequence of cities (usually minimum cost, distance, or time) in which to complete the tour. The TSP can be viewed as a graph theory problem if the cities are identified with nodes of a graph and the 'links' between the cities are identified with the arcs.[1]

It is not known who brought the name TSP into mathematical circles, but there is no question that Merrill Flood is most responsible for publicizing it within that community and the operations research fraternity as well. He recalls being told about it by A.W. Tucker in 1937 'when I was struggling with the problem in connection with a school-bus routing study in New Jersey.' Flood writes that Tucker remembered hearing of it from Hassler Whitney at Princeton

This thesis follows the format of the *Journal of the Operational Research Society.*

University. This would seem to make Whitney the entrepreneur of the TSP, but Whitney remembers no connection whatever between himself and the TSP.[6]

Despite the confusion behind its origin, the TSP can be said to formally have a name starting at least as early as the late 1940's. John Williams urged Flood in 1948 to popularize the TSP at the RAND Corporation, at least partly motivated by the purpose of creating intellectual challenges for models outside the theory of games.[6] In 1954, Dantzig, Fulkerson and Johnson published a landmark paper entitled 'Solution of a large scale traveling-salesman problem' in the Journal of the Operations Research Society of America. In this paper, the authors not only solved a large TSP, but introduced the concept of cutting planes for use in integer linear programs and may have been the first to introduce the technique of branch and bounding as a tool for solving combinatorial optimization problems.

In 1963, Little et al.[8] published a paper detailing a branch-and-bound algorithm which could solve a 13 city problem by hand in under 4 hours. This algorithm could also easily be written in computer code. Using an IBM 7090 computer, problems up to 20 cities could be solved in seconds, however the solution time grew exponentially and by 40 cities the computation time exceeded 8 minutes. In the early 1960's, the TSP achieved national prominence when a soap company used it as the basis of a promotional contest. Prizes up to $10,000 were offered for identifying the most correct links in a particular 33-city problem. Quite a few people found the best tour. A number of people, perhaps a little over-educated, wrote the company that the problem was impossible - an interesting misinterpretation of the state of the art.[6]

Since the 1960's, much of the research on the TSP has dealt with finding algorithms and heuristic with solve increasingly larger problems. Today, it is generally accepted that Little's algorithm works well for problems with less than 50 cities. In 1980, Crowder and Padberg[9] presented a solution to a 318 city symmetric TSP. The solution used 0-1 integer programming, and was solved in under 6 minutes of CPU time on an IBM 376/168 computer. In 1984, Kirkpatrick reported on a problem containing 6,406 cities, however whether or not the problem was solved was unknown[6]. As of 1985, the 318 city problem was the largest problem known to have been solved. Since then heuristic approaches such as the Out of Kilter Algorithm have been applied to many larger problems as a way of determining optimal or near optimal solutions in a reasonable amount of computing time. Finding heuristic and optimal solutions to problems larger than 1000 cities is still the focus of much of the current research on this problem.

Problem Introduction

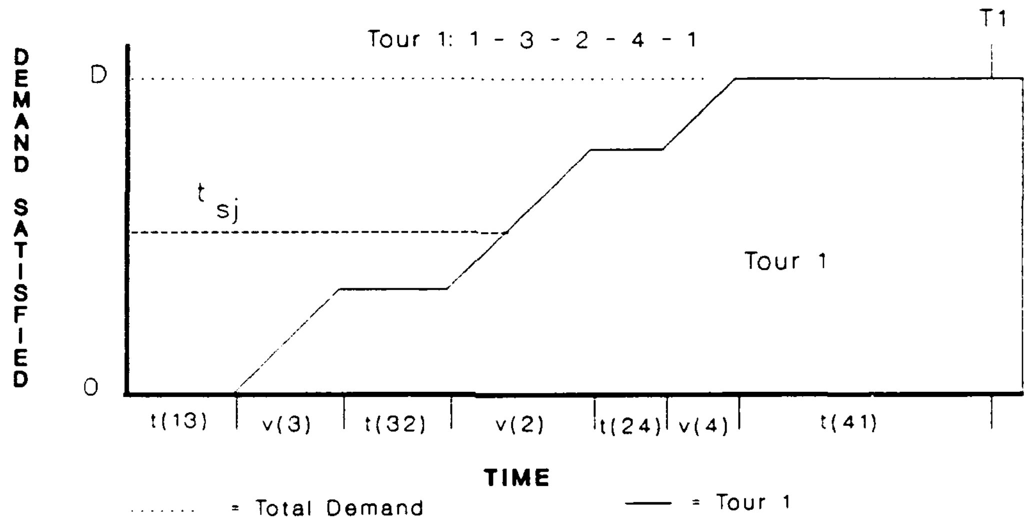
The classical TSP has been used in a wide range of applications. In addition, there have been numerous modifications made to the classical TSP which broaden its applications. Some examples of modifications to the TSP include multi-salesman TSP, the time sensitive vehicle routing problem[4], and the vehicle routing problem with distance and capacity constraints.[7]

In an interesting modification to the TSP, the salesman begins at a home city and visits a set of cities with specified quantities of demand that need to satisfied. When the salesman arrives at city j, he services the demand d_j at a given service rate r_j of units per time unit. Upon satisfying every unit of demand at that city, the salesman proceeds on to the next city and continues in this manner until all cities are visited and the corresponding demands are satisfied.

The salesman in this problem could have one of several different objectives. One objective for this problem could be to satisfy the entire demand in the shortest possible time. In this situation, the length of the return trip from the last city is unimportant since all of the demand has already been satisfied. This problem can be easily solved as a classical TSP by setting the lengths of the arcs going from each city to the home city to zero. The solution to this problem provides the shortest path to the last city while ignoring the demand altogether and the return trip from the last city.

A more complex problem arises when the objective is to minimize the average time required to satisfy each unit of demand. This problem is complicated because the time to satisfaction for any unit of demand is dependent upon the times to satisfaction for all previous satisfied units. This problem can be viewed graphically by plotting the demand satisfied versus time as shown in Figure 1. Any feasible tour can be plotted in this manner, with each tour having a unique demand satisfaction function. The times required to satisfy each unit of demand make up the area to the left of the demand satisfaction function and below the total demand line.

In this figure, the horizontal lines of the function represent the time required to travel between cities t_{ij}, and the diagonal lines represent the service times while demand is being satisfied within a city. The service time at each



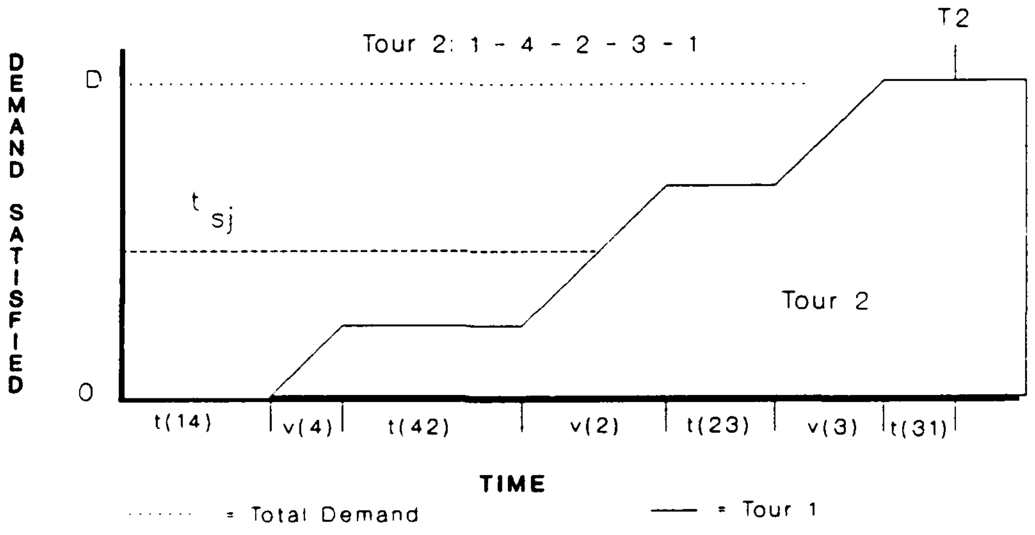


Figure 1 Demand Satisfaction Functions

city v_j can be calculated as follows:

$$v_j = \frac{d_j}{r_j} \tag{1}$$

If t_{sj} is the time until the j^{th} unit of demand is satisfied and D is the total number of units being demanded, then the average time to demand satisfaction can be represented by the following relationship:

$$Avg.\ Dem.\ Sat.\ Time = \sum_{j=1}^{D} \frac{t_j}{D} \tag{2}$$

From Equation (2), it should be clear that the average time to demand satisfaction is a function of the area to the left of the satisfaction curve and below the total demand line as seen in Figure 1. Therefore the tour which minimizes this area will also minimize the average time to satisfaction. In addition, a comparison of Tours 1 and 2 in Figure 1 illustrates that the shortest path does not always represent the lowest average time to satisfaction. While Tour 2 is completed prior to Tour 1, Tour 1 clearly has a lower average time to demand satisfaction (smaller area above the satisfaction curve).

This type of problem can be applied to many service related activities, where customers are serviced from a depot location. As an illustration of this type of problem, consider an aircraft maintenance depot. Assume there is a type of aircraft with a faulty component which can cause a safety problem. Further assume that only a single team of experts are qualified to make the modifications that alleviate the problem. These aircraft are assigned to various locations and in varying quantities. Because it is important to maintain this fleet of aircraft in an operational capacity, a route must be chosen for the depot modification team which will modify the aircraft to achieve the highest operational capacity.[2] This can only be done by minimizing the average time to aircraft modification completion over the entire fleet.

Another real world example can be illustrated in the manufacturing environment. Assume that a team of technicians are required to modify the tooling on machines and train machine operators for the production of a new product which the company has just entered into contract. The company has the machines required for this product located in different plant facilities and in various quantities. After each machine is modified, and its operator trained, it can begin producing the new product and thereby generate revenue for the company. Therefore, in order to maximize the revenue generated, a tour must be found through the facilities which minimizes the average time to modification completion over all the machines in the company.

Research Objectives

The objective of this research is to develop an algorithm which finds the optimal solution to the problem described above. The proposed solution approach uses a branch-and-bound procedure and does not require the complete enumeration of all feasible solutions. A computer code will be developed to implement the proposed solution procedure. The final product of this research will be a decision-support methodology that can be used to determine the most attractive sequence of cities given the travel times between cities and the demand and service rate at each city.

Literature Review

During this research effort, the following literature was reviewed in order to determine if any previous work had been accomplished on this problem and for assistance in the development of the solution procedure presented in this paper.

Arthur and Frendewey[1] presented an algorithm for the generation of TSPs with known optimal tours. This algorithm has no limitation on the number of nodes which can be input into the problem. This algorithm is useful for the creation of large TSPs for the design and testing of alternative TSP solution approaches.

Coutois, Grant, and Kane[2] prepared a project report at Texas A&M which evaluated different approaches for solving a problem nearly identical to the problem in this research effort. Major differences in their problem were the instantaneous satisfaction of demand upon arrival at a city (as opposed to satisfying demand at a given service rate) and allowing more than one station (demand source) at each city. Of the five solution approaches reviewed in this report, the only approach which could guarantee optimality was complete enumeration. Due to the combinatorial nature of this approach, the computer code generated to implement this approach was limited to small problems (6 or less nodes).

Phillips and Garcia-Diaz[3] presented an in-depth description of Little's branch-and-bound algorithm. This algorithm is used to solve the classical TSP for shortest tour through a set of nodes. Along with this description was an explanation of the computational procedures required to implement this algorithm. This algorithm was evaluated in this research as a tool in identifying an initial upper bound.

Evans and Norback[4] developed a heuristic for solving time sensitive routing problems, using a time series density function which identifies clusters of stops to form a route. This problem encompassed the idea of demand at different locations. However, this routing problem differs from the TSP in that more than one sub-tour may be generated to complete the tour of nodes. These sub-tours can then be assigned to any number of vehicles. Also the time constraint in this problem was a window of time in which demand needed to be satisfied.

Padberg and Rinaldi[5] describe a branch-and-cut algorithm for solving large symmetric TSPs. This algorithm differs from the branch-and-bound in that whenever the cutting plane procedure does not terminate with an optimal solution, the algorithm uses a tree search strategy that, as opposed to branch-and-bound, keeps on producing cuts after branching. Since this algorithm works only for symmetric TSPs, it was not able to handle the problem at hand.

Lawler et al.[6] edit a book which looks in detail at the origin, history, applications, and solution approaches to many forms of the TSP. While this book does describe many variations to the classical TSP, no discussion was found of a problem similar to the problem in this research effort.

Laporte, Nobert, and Desrochers[7] present a branch-and-bound algorithm for vehicle routing involving capacity and distance constraints. This algorithm can be used to solve for symmetrical routing problems only and also allows for the generation of sub-tours.

Little et al.[8] describe a branch-and-bound algorithm for solving asymmetric TSPs with sizes between 8 and 50 cities. This algorithm is well known for its computational ease and its ability to be easily implemented with computer code.

Crowder and Padberg[9] presented an algorithm for the solution of large symmetric TSPs. This algorithm used 0-1 integer programming to solve problems of up to 318 cities.

This review of literature did not find any solution to the form of TSP presented in this research (with the exception of complete enumeration, Courtois et al.[2]). Since this solution approach is impractical for large problems, a solution approach developed not requiring complete enumeration would significantly reduce the time required to solve many medium to large size problems.

Thesis Organization

This thesis is organized into five chapters which present the results of the research in the following manner:

Chapter I: Introduction - This chapter provides an introduction to the Travelling Salesman Problem (TSP) and the particular application of the TSP to be discussed in this paper. This chapter also describes the goals of this research, the literature review conducted, and the organization of this thesis.

Chapter II: Problem Development - This chapter provides an in-depth description of the problem and shows the graphical representation of the problem. It also provides a problem definition and the operations needed to initialize the input data in a form for use by the solution approach.

Chapter III: Solution Approach - This chapter provides development of the solution approach using a branch-and-bound algorithm. The procedures necessary for initial upper bound generation, the decision methodology for choosing the subsequent branch to search, and lower bound generation at any branch are presented in detail.

Chapter IV: Application of Solution Procedure - This chapter demonstrates the implementation of the solution procedure developed in Chapter III using two examples of different size.

Chapter V: Results, Conclusions, and Recomendations - This chapter describes the results of this research and the conclusions drawn from this effort. It also describes the lessons learned and makes recommendations for future work which can be conducted in this area.

CHAPTER II

PROBLEM FORMULATION

This problem is probably best described in a graphical context. As shown in Figure 1, the tour which produces the smallest area above the demand satisfaction curve and below the total demand line also produces the minimal average time to demand satisfaction. Therefore this problem can be viewed as to finding the tour which minimizes this area.

As seen in Figure 2, the shaded area represents those times when demand is being satisfied at each node. Since any feasible tour must satisfy demand at all nodes, these shaded areas will be included in all feasible tours and thus can be removed from consideration in the optimization problem. This reduces the demand satisfaction function to a step function in which each rectangle (i.e. A,B and C) represents the decision to include a particular arc (i,j) in a specific sequence position in the tour. It should also be clear that if a decision is made to include arc (i,j) in a tour, while the width (horizontal) of the rectangular area for this arc will be a constant determined by the distance matrix $\mathbf{T} = [t_{ij}]$ and the service time v_i, the length (vertical) of the rectangle will depend upon the sequence position of the arc (i,j) and which arcs have preceded it.

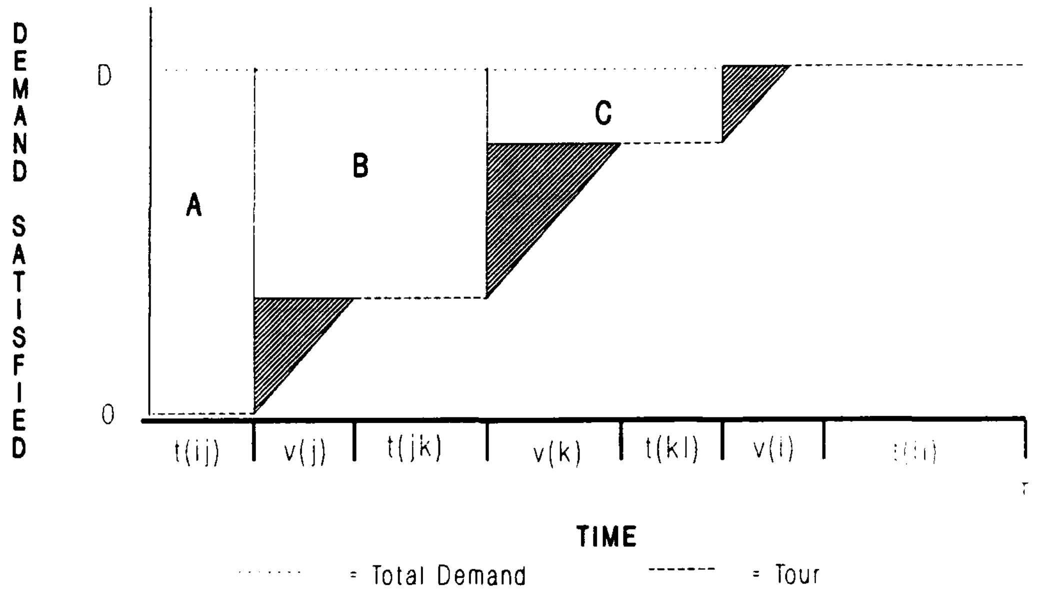


Figure 2 Demand Satisfaction Step Function

The fact that each rectangular area depends upon the set of arcs previously visited makes this problem dynamic in nature. Understanding this dynamic property of the problem is fundamental to the development of a solution approach. There currently exist algorithms which can minimize the length of the tour (the horizontal component of the rectangular areas), however as the length of the tour is reduced the total areas may either decrease or increase depending on the sequencing of the nodes in the subsequent tour. It is for this reason that classical traveling salesman algorithms will not optimize this problem for minimum average time to satisfaction. This dynamic property also prevents the use of penalty assessment for not using a certain arc (as in Little's Algorithm) and forces any solution procedure to check as a minimum each subset of tours defined by every arc (n-1 subsets) which originates at the source (first) node.

Problem Definition

The problem can now be defined as follows: Given a network **G** = (**N**,**A**), where **N** is a set of nodes representing cities and **A** is the set of arcs (i,j) representing the travel time between cities i and j. Also given **D** = (d_j), where d_j represents the unsatisfied demand at city j and **R** = (r_j), where r_j is the rate at which demand is serviced at any city. Find the tour through **N** which minimizes the average time to satisfaction for **D**. If the set of all previously visited nodes at any point in the tour is defined as **B**, the problem can then be described as follows:

$$\textit{Min}: \sum_{i=1}^{n}\sum_{j=1}^{n} \lambda_{ij}(t_{ij} + s_i)U_i \qquad (3)$$

$$s.t. \sum_{j=1}^{n} \lambda_{ij} = 1 \quad \forall\, i \qquad (3\text{-a})$$

$$U_i = D - \sum_{j \epsilon B} d_j \tag{3-b}$$

$$\lambda_{ij} = (0,1) \qquad \forall\, ij \tag{3-c}$$

In these equations, λ_{ij} represents an integer, 1 if arc (i,j) is included in the tour, and 0 if arc (i,j) is not included in the tour. The objective function for the total area is Equation 3. Equation 3-a is the constraint which guarantees a tour through **N** without subtours, Equation 3-b is the definition of U_i, and Equation 3-c defines λ_{ij} as a (0,1) integer variable.

Input Data Requirements

In order to use the solution procedure presented in the next chapter, the following data are required: 1) a distance matrix **T** = $[t_{ij}]$ which provides defines the travel time between each city in **N**, 2)the set of demands **D** = (d_j) which defines the demand in units at each city in **N**, and 3) the set of service rates **R** = (r_j) which defines the rate of satisfaction demand satisfaction in units per time unit for each city in **N**. This input data can now be initialized into a form to be used in the solution procedure that follows. The distance matrix **T** is shown below as it would be input into the solution procedure.

$$T = \left[t_{ij}\right] = \begin{bmatrix} \text{—} & t_{12} & t_{13} & t_{14} \\ t_{21} & \text{—} & t_{23} & t_{24} \\ t_{31} & t_{32} & \text{—} & t_{34} \\ t_{41} & t_{42} & t_{43} & \text{—} \end{bmatrix}$$

The first step is to set all arcs (i,1) = 0 in the time matrix. This is done because the length of the return trip from the last city visited is unimportant since all demand is already satisfied. The next step is to add v_i to all arcs (i,j). This is done because, as shown in Figure 2, the service time v_i added to the arc (i,j) provides the width of the rectangle associated with including arc (i,j) in the tour. It should noted that for the solution procedure presented in this thesis, it is assumed that node 1 is the node of origin at that it has a demand of 0 units. Therefore when using the solution procedure $v_1 = 0$. The resulting matrix is shown below.

The third step in the initialization of the input data is to create a slope matrix **S** = $[s_{ij}]$ by dividing each element in the new time matrix t'_{ij} by the demand at node j, d_j, as shown below.

$$T' = \left[t'_{ij}\right] = \begin{bmatrix} - & t_{12} & t_{13} & t_{14} \\ 0 & - & t_{23}+v_2 & t_{24}+v_2 \\ 0 & t_{32}+v_3 & - & t_{34}+v_3 \\ 0 & t_{42}+v_4 & t_{43}+v_4 & - \end{bmatrix}$$

$$S = \left[s_{ij}\right] = \begin{bmatrix} - & t'_{12}/d_2 & t'_{13}/d_3 & t'_{14}/d_4 \\ 0 & - & t'_{23}/d_3 & t'_{24}/d_4 \\ 0 & t'_{32}/d_2 & - & t'_{34}/d_4 \\ 0 & t'_{42}/d_2 & t'_{43}/d_3 & - \end{bmatrix}$$

The values or slopes in this matrix represent the demand satisfaction rate associated with any arc (i,j). The graphical representation of the slopes is shown in Figure 3. This matrix will become useful in the development of the solution procedure. It should be clear that a large slope appears to play a role in achieving a low average time to satisfaction. An arc with a larger slope can reduce area of the associated rectangle by satisfying the same amount of demand at a greater rate (reducing the width of rectangle A), or reduce the area of the subsequent rectangle by satisfying a greater demand in the same amount of time and thus decreasing U_i (reducing the length of B).

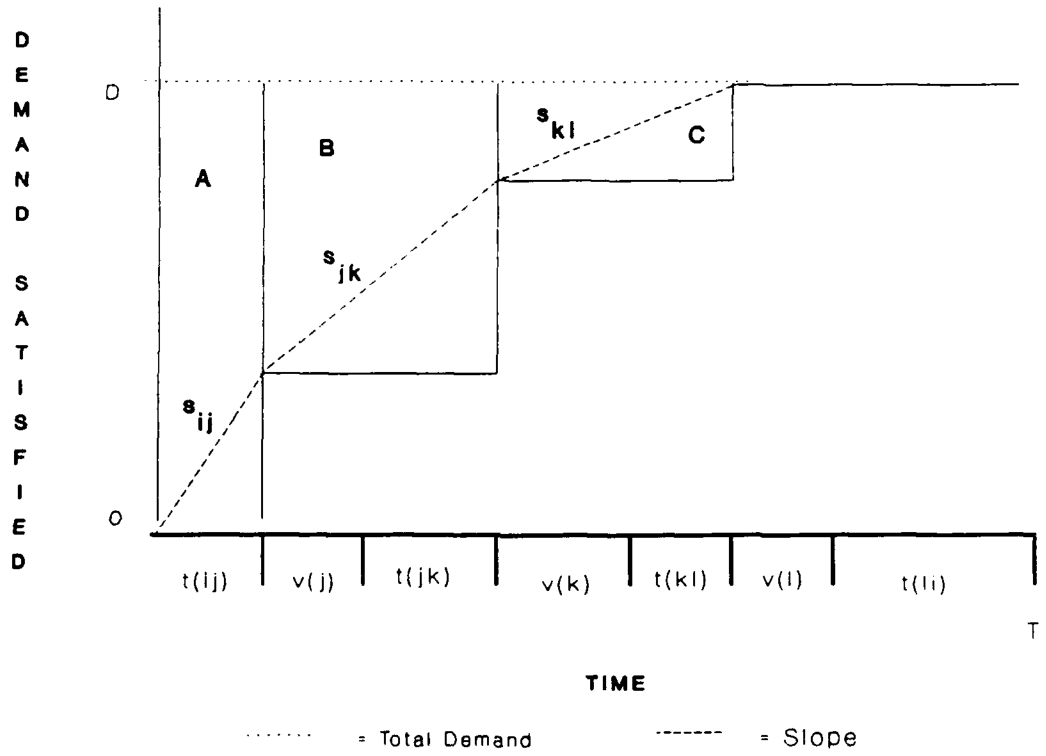


Figure 3 Graphical Representation of Slopes

The input data are now in a usable form for the solution procedure discussed in Chapter III.

CHAPTER III

SOLUTION APPROACH

The algorithm developed in this thesis uses a branch-and-bound approach find the optimal tour. Any branch-and-bound approach contains the three following elements:

1) a procedure for the generation of an initial upper bound,
2) selection criteria for selecting the next node to search, and
3) a procedure for the generation of a lower bound for any selected node (or subset of tours from the set of all feasible tours).

Upper Bound Generation

It is important to find a technique which identifies a good initial upper bound for the solution space. The effectiveness of fathoming strongly depends on the goodness of the starting solution. In fact, the lower the value of the upper bound, the smaller the number of nodes in the search tree.[5]

The slope matrix becomes a valuable tool in the development of a procedure for determining an upper bound. This matrix provides the rate of demand satisfaction for each arc in the time matrix. Any good initial upper bound should have high rates of satisfaction and should place greater emphasis on arcs traversed earlier in the tour because these arcs have a greater impact on the total area than those towards the end of the tour.

Three approaches to finding an upper bound were evaluated during this research. The first approach was to find the tour which contained slopes that summed to the greatest value. This could be accomplished relatively easily by multiplying the slope matrix by -1 and solving for the shortest tour using Little's Algorithm. The advantage of this approach is that it uses a well known algorithm and it views the whole tour in terms of maximizing the rate of demand satisfaction, thus taking a "global view" of the problem. The disadvantage to this approach is that it does not consider the order in which the slopes are sequenced into the tour. In other words, this approach weights the value of a high slope at the end of the tour equally with a high slope in the first arc traversed.

The second approach evaluated was a type of greedy algorithm. In this approach, the largest slope leaving the first node (row 1) is chosen as the first arc (i,j) in the tour. Row i and column j are then deleted from the matrix and the highest slope in row j is chosen as the next arc (j,k) in the tour. This procedure continues until the tour is completed. The advantage to this algorithm is that it is easy to implement and it considers the order in which the arcs are entered into the tour. The primary disadvantage of this approach is that while it maximizes the rate for any one leg of the tour, it does not consider the impact of that choice on any future legs. Therefore this approach can easily be "turned away" from a good initial upper bound by one high slope early in the tour.

The third approach evaluated is a two-step greedy algorithm. This approach tries to incorporate some of the "global view" of the first approach with the "importance in order" of the second approach. In this approach, the slope for each arc (1,j) is added to the maximum slope for arc (j,k) where k is not equal to 1 or j. Whichever two arcs yield the highest sum of slopes are the arcs chosen for the first two arcs in the tour. This procedure is then repeated starting in row k, and continues until the tour is completed. The advantage of this approach is that it places importance on the order in which the arcs are entered into the tour, and by selecting two arcs at one time, provides a more global view of the problem than the greedy algorithm. The flow diagram for the two-step greedy algorithm is shown in Figure 4.

Selection Criteria for Search

All branch-and-bound algorithms are based upon on the idea of a search tree made up of nodes and branches. Search trees for traveling salesman problems can be viewed as shown in Figure 5. These search trees for begin at a source node which represents the set of all feasible tours. Each node in level 1 represents a subset of the source node. Subsequently, each node in level 2 represents a subset of a node in level 1 with the branches identifying the subset relationship. In TSPs, each level represents the decision to include an arc in the tour. Hence a feasible tour can be uniquely defined in (n-1) levels. Therefore the maximum of nodes will be:

$$\sum_{i=1}^{n-1} \frac{(n-1)!}{(n-i)!} \tag{4}$$

As previously stated, the dynamic nature of this problem will ensure that, as a minimum, all nodes in level 1 of the search tree must be searched to ensure optimality. Thus the

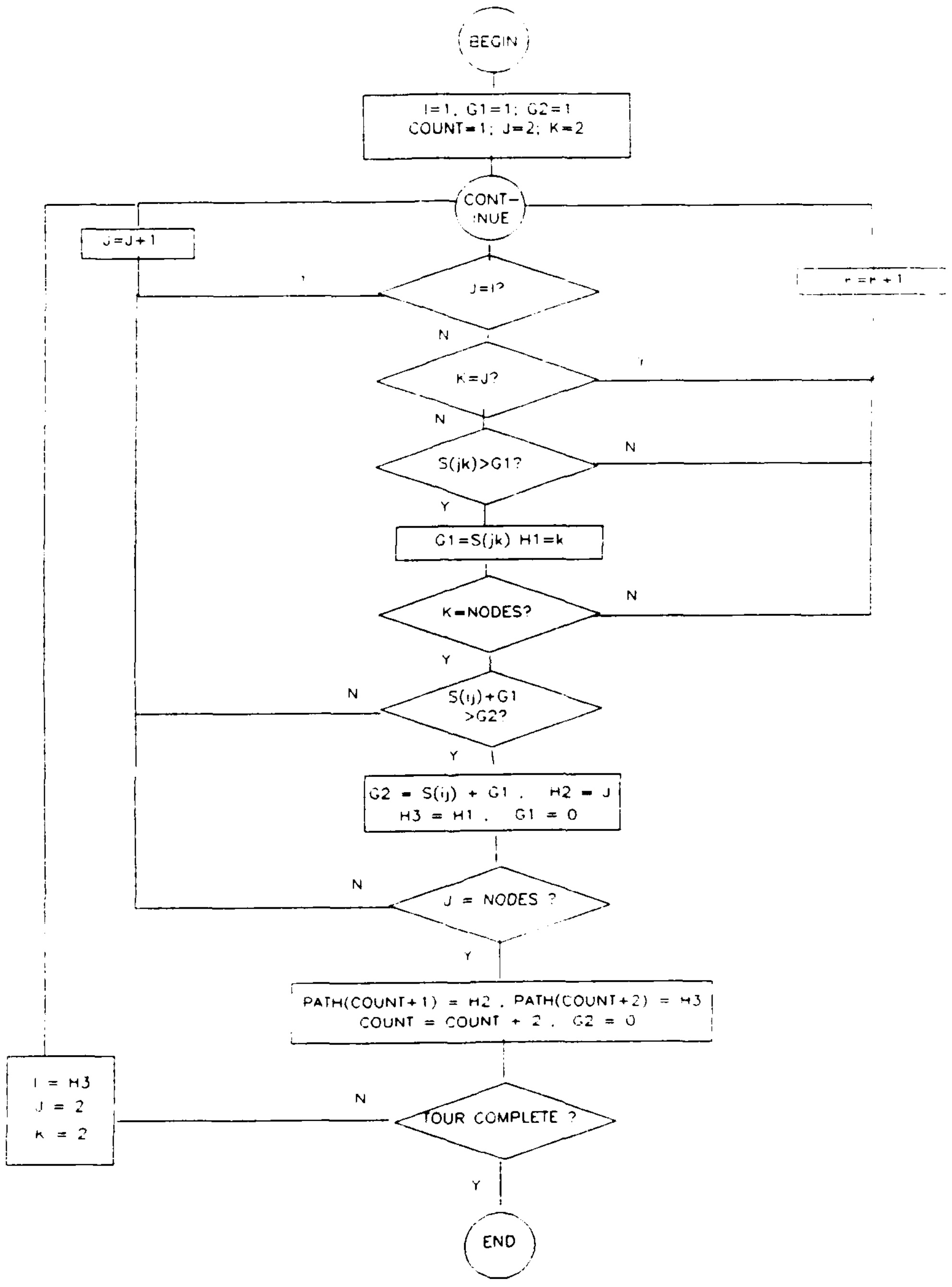


Figure 4 Two-Step Greedy Algorithm Flowchart

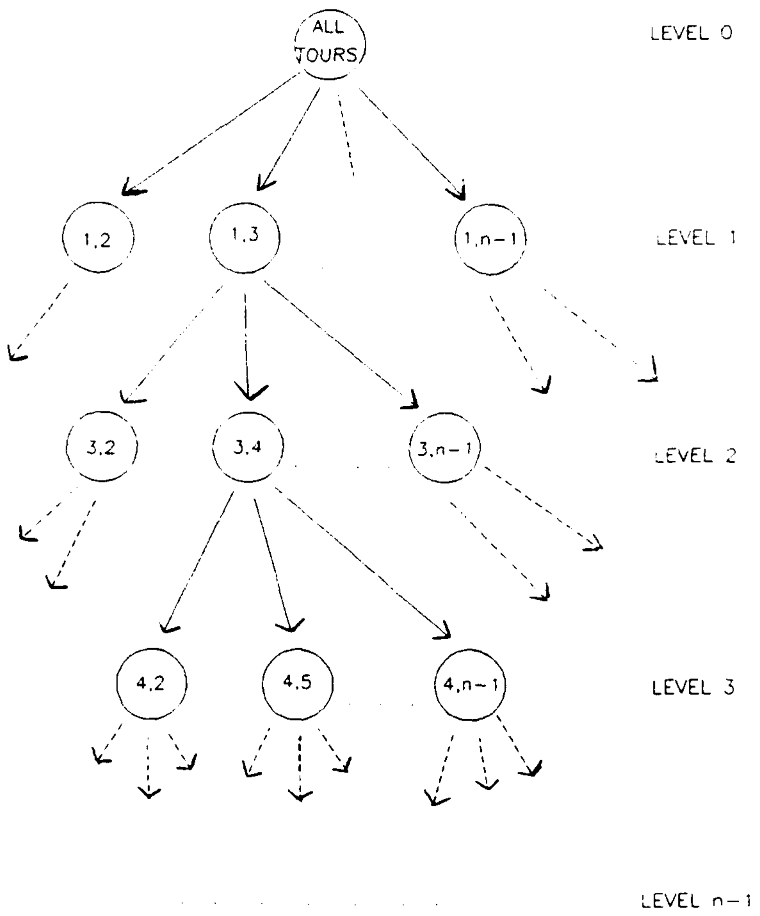


Figure 5 Search Tree

objective of the search criteria should be to minimize the number of nodes searched in the tree. This can be accomplished through fathoming. When a node (subset of tours) has a lower bound greater than the existing upper bound, that entire subset of tours need no longer be searched and the node is said to be fathomed. Therefore the selection criteria for the next node to search should identify a node which will be likely to lead to a tour with a lower bound less than the value of the existing upper bound. If such a tour is found, the upper bound will then take on the value of that tours lower bound. By reducing the upper bound, the process of fathoming is facilitated for all subsequent nodes searched.

The criteria and procedure for selecting the next node to search in the tree can be described as follows:

Step 1. Optimality Check.

If the largest slope equals 0 and the level is 1, then the current upper bound is the optimal solution. Stop.

Step 2. Node Selection.

Choose to search the node associated with the largest slope in the current level which does not create a subtour. If largest slope equals 0, then back up one level and go to Step 1, otherwise go to Step 3.

Step 3. Lower Bound Calculation.

Calculate the lower bound for the node chosen using the procedure described in the following section.

Step 4. Check for Fathoming.

If the lower bound calculated is greater than the existing upper bound, the node is fathomed, the slope of the arc associated with it is set to zero, back up one level, and go to step 1.

Step 5. Continue Search.

If the lower bound is less than the existing upper bound and the level is (n-1), then the upper bound is set equal to the lower bound, the slope of the arc associated with that node is set to 0, back up one level and go to step 2. Otherwise, if the lower bound is less than the upper bound and the level is not (n-1), then go forward one level and go to step 2.

Lower Bound Generation

This task is probably the most important and most difficult step in developing a branch-and-bound algorithm. The primary purpose of generating a lower bound is to group the solution space into subsets of possible solutions and bound each subset by a minimum value for which any solution in the subset must exceed. A good lower bound generator has three basic properties:

1) a mathematical basis for bounding the subset (the generator will bound any subset in all situations).

2) relative ease of calculation (the generator should not require any form of enumeration of the

subset solutions).

3) the estimate of the lower bound should not be too conservative (too conservative estimates hinder the process of fathoming).

Based upon the previous three stated properties, the following procedure was developed to generate a lower bound for any node (subset of tours, **H**, defined by arcs) in the search tree:

Step 1: For all arcs (i,j) in the subset **H**,

$$LB_1 = \sum_{(i,j)\epsilon H} t'_{ij} U_i \qquad (5)$$

where:

LB_1 = the total area of all arcs in the current subset

t'_{ij} = the initialized matrix distance for arc (i,j)

U_i = the unsatisfied demand upon departing node i

The graphical representation of LB_1 can be seen in Figure 6. LB_1 can be seen as the sum of the areas of rectangles A and B.

Step 2: As can also be seen in Figure 6, the unsatisfied demand U_i, for any subsequent arc, has already been defined by the arcs currently in the tour. Therefore the minimum area the subsequent rectangle, LB_2, can be defined as:

$$LB_2 = U_i[\min_j (t'_{ij})] \tag{6}$$

where i is the terminal node of the last arc in the current tour, and j is any node (city) which will not complete a subtour. LB_2 can be seen graphically in Figure 6 as the area of rectangle C. To avoid completing subtours, as an arc (i,j) enters the tour row i and column j are set to infinity (lined out) in the **T'** matrix.

Step 3: In order to ensure that the remaining area is bounded, it shall be assumed that the largest remaining demand d_j is satisfied at the largest remaining rate s_{ij} in **S** as shown in Figure 6. The next largest demand would then be satisfied by the next largest rate and so on until there is no unsatisfied demand. As shown graphically in Figure 7, the max slope and max demand define the width of the subsequent rectangle by:

$$width = \frac{\max\ demand}{\max\ slope} \tag{7}$$

The area of the rectangle is calculated by multiplying the width by the present unsatisfied demand U_i. The areas for subsequent rectangles can be calculated by updating U_i (by subtracting max demand), max demand (by removing max demand and finding next largest), and max slope (same as max demand). The sum of the areas calculated in this manner LB_3,

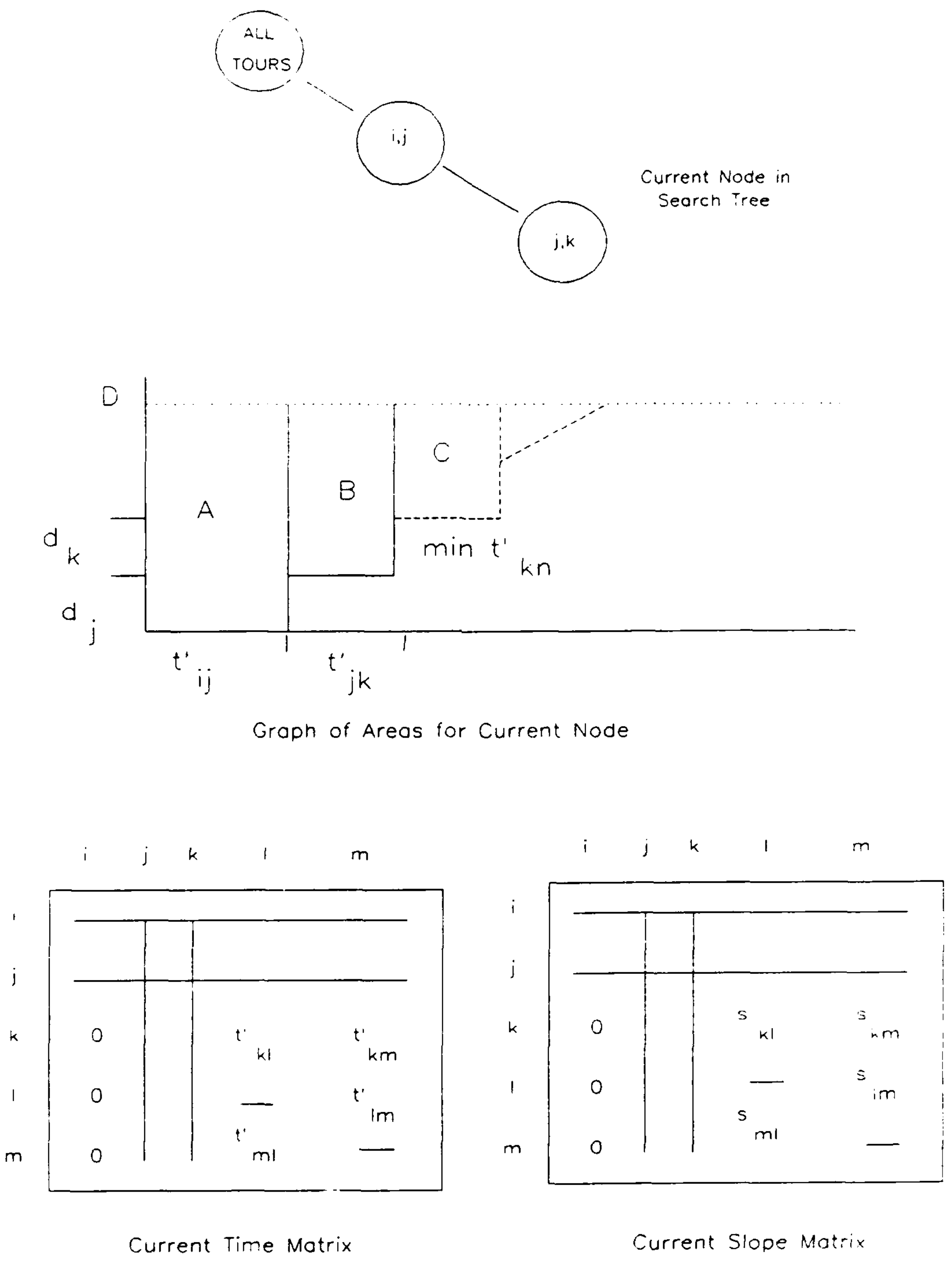


Figure 6 Graphical Representation of Lower Bound Calculation

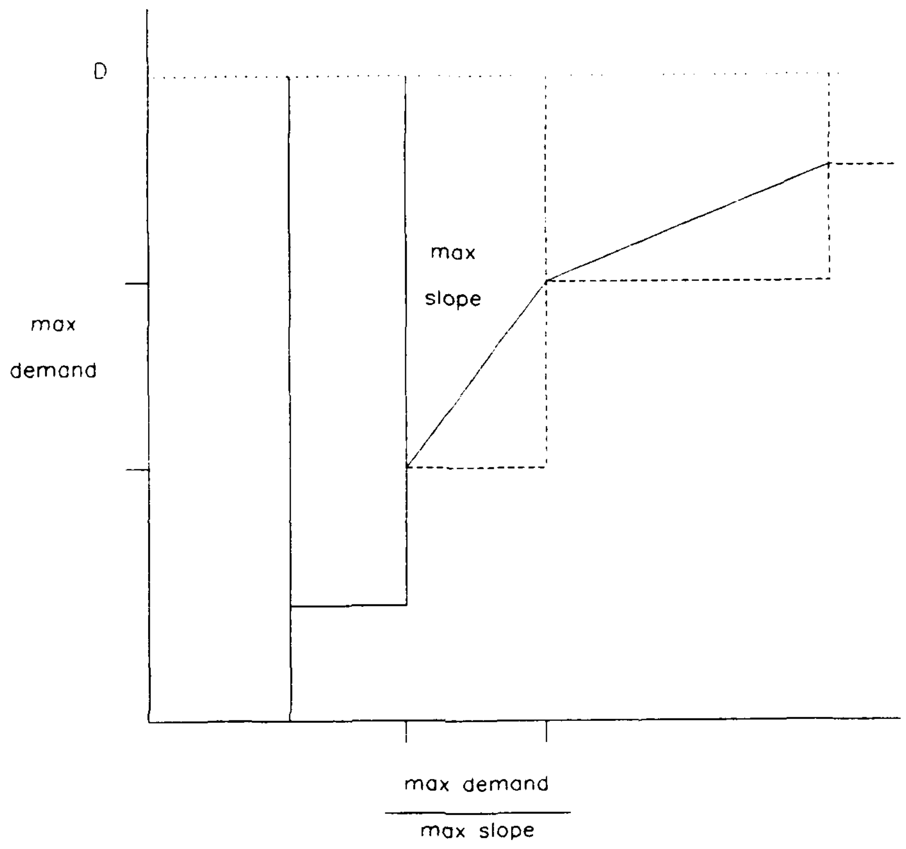


Figure 7 Use of Slopes for Lower Bound Calculation

demand). The sum of the areas calculated in this manner LB_3, can be calculated as:

$$LB_3 = \sum \left(\frac{\max \ demand}{\max \ slope}\right) U_i \tag{8}$$

And the lower bound for the present search node can now be defined as:

Once the lower bound is calculated for any node, it can

$$LB = LB_1 + LB_2 + LB_3 \tag{9}$$

Once the lower bound is calculated for any node, it can be compared to the existing upper bound as described in the previous section to determine if the node can be fathomed, the search continue, or if a better solution has been found.

CHAPTER IV

APPLICATIONS OF SOLUTION PROCEDURE

The solution approach presented in Chapter III was coded in GW-Basic, as shown in Appendix A, so that its performance

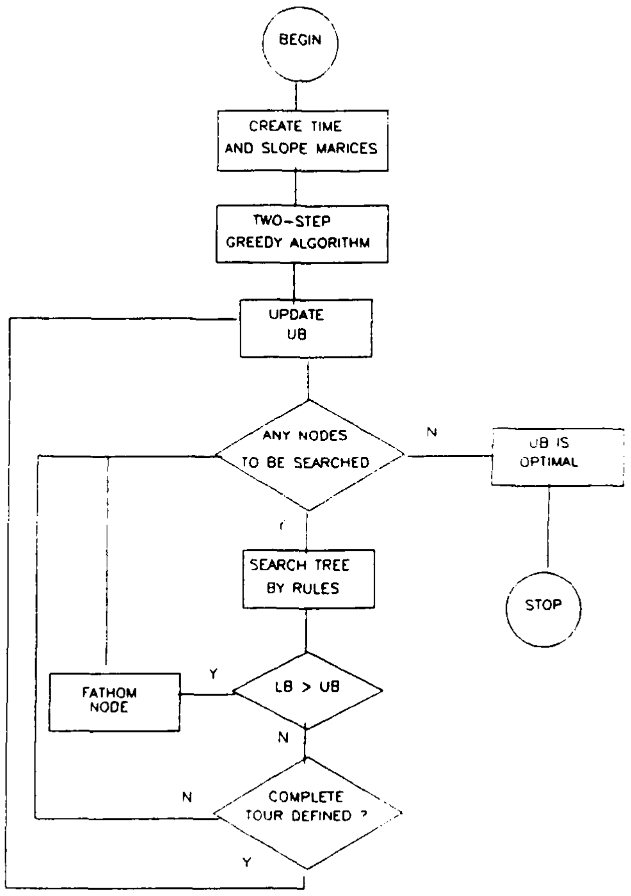


Figure 8 Program Flow Chart

could be evaluated. The flow chart describing the basic logic behind the program can be seen in Figure 8. The User Instructions for this code are presented in Appendix B. It should be noted that the program presented was designed for an asymmetrical TSP with the number of nodes being between 3 and 20. The program always assumes that node 1 is the source (home) and has a demand of 0. The program begins after the user enters the data as described in the User Instructions. The program initializes the time matrix and creates the slope matrix. Next, the two-step greedy algorithm is performed and the initial upper bound and its path are found. The tree search is then carried out in accordance to the decision rules described in Chapter III. If a tour is completely defined in the search tree, and it has a lower bound less than the existing upper bound, the existing upper bound is replaced by this lower bound. At the conclusion of the tree search, the existing upper bound will be the optimal solution and the tour which defines that upper bound will be the optimal tour.

Two examples of different size were chosen to validate the solution approach with the computer program. The first example contains six cities and assumes that the service rate is the same for each city in **N**. The second example shows the ability of the solution procedure to handle a larger problem

(11 cities) and a varying service rate between cities. The first example problem used for validation is described below.

Example 1

Given the matrix **T** of travel times between the set of 6 cities **N**, the set of corresponding demands **D**, and a of service rates 0.5 units/hr; find the tour through **N** which minimizes the average time to demand satisfaction.

$$T = \begin{bmatrix} -- & 25 & 50 & 20 & 35 & 29 \\ 23 & - & 20 & 36 & 25 & 54 \\ 48 & 18 & -- & 33 & 41 & 44 \\ 18 & 24 & 37 & -- & 22 & 22 \\ 37 & 43 & 51 & 26 & -- & 54 \\ 28 & 33 & 48 & 40 & 50 & - \end{bmatrix} \qquad D = \begin{bmatrix} 0 \\ 18 \\ 23 \\ 25 \\ 29 \\ 32 \end{bmatrix} \qquad R = \begin{bmatrix} 0 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix}$$

Figures 9-11 show examples of the search path used by the solution approach and the lower bound values calculated at each node. Figure 9 shows the minimum number of nodes which must be searched in a 6 node problem. Since none of the nodes searched can be fathomed, the next level in the tree must be searched with each of these nodes as the source of the branches.

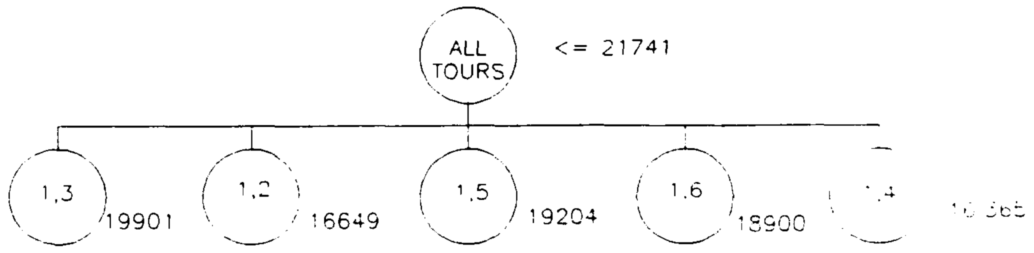


Figure 9 Minimum (n-1) Nodes Searched

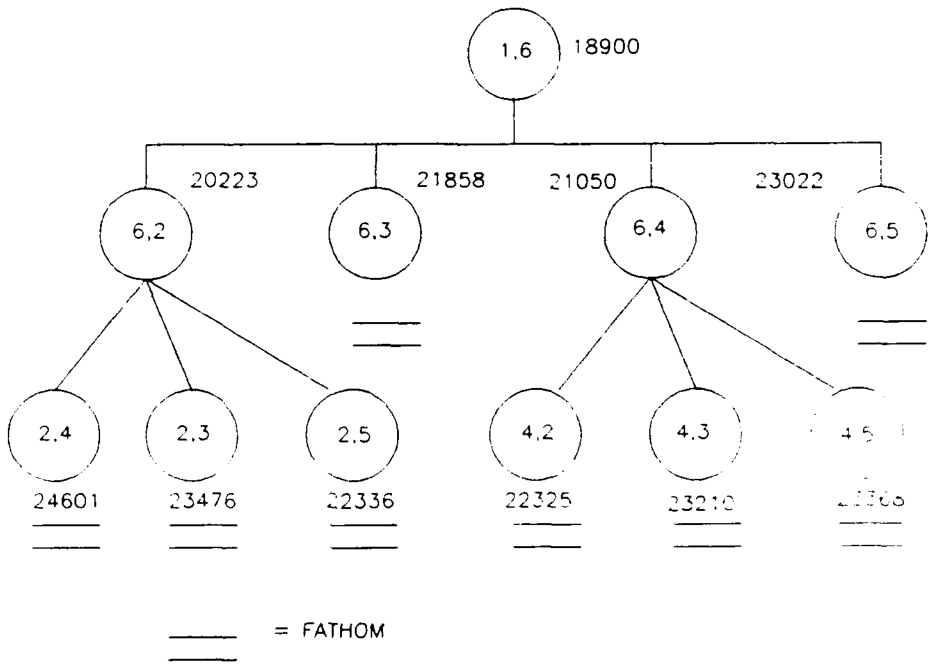


Figure 10 Tree Search - Source = (1,6)

For Figures 9-11, the solution procedure starts at the top node and always proceeds down and from right to left. For example, as seen in Figure 9, the first subset of tours

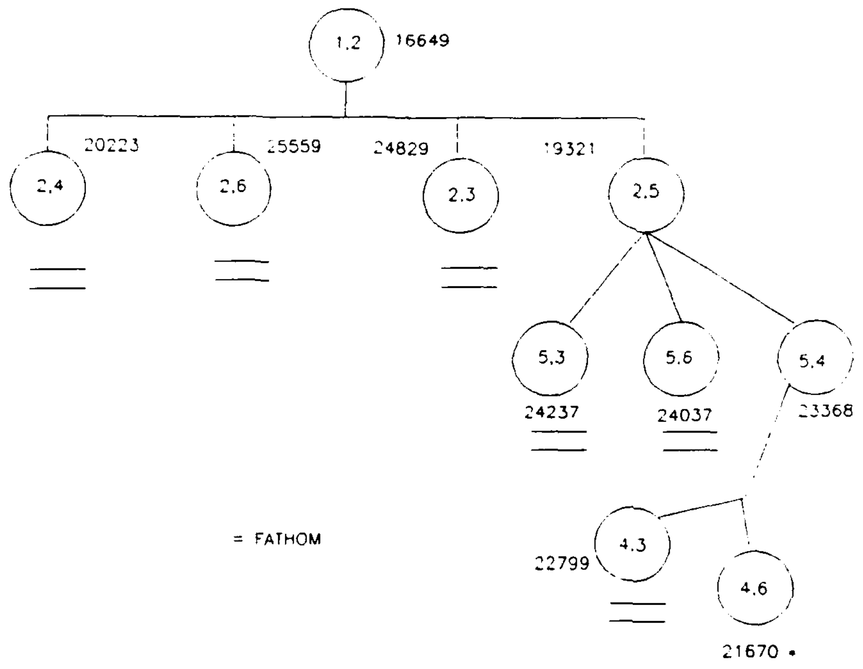


Figure 11 Tree Search - Source = (1,2)

to be searched were those containing arc (1,4). Since the lower bound for this subset was less than the existing upper bound (from two-step greedy algorithm), the search continued downward until all nodes below (1,4) were fathomed. The search then proceeded on to the subset of tours containing arc (1,6) as shown in Figure 10. Of the 40 possible nodes which define this subset of tours, only 10 were required to be searched before the entire subset could be fathomed. Since there are 24 possible tours which include arc (1,6) solving this problem by enumeration would require the calculation of all 24 total areas before it could be determined that this subset does not contain an optimal solution. Using the solution procedure presented in Chapter III, the calculation of only 10 areas was required. This clearly shows the value of the branch-and-bound procedure developed in this thesis.

Figure 11 shows the search conducted of the subset of tours containing arc (1,2). In this figure, it can be seen that the solution procedure finds a lower bound (21,670) which defines a complete tour and is lower than the existing upper bound (21,741). In this case, the upper bound is now set to 21670 and the search continues in this manner until all nodes in the search tree are fathomed or found to have lower bounds greater than the existing upper bound. At this point the upper bound is the optimal solution.

The optimal solution was found to be the tour 1-2-5-4-6-3-1, with an area of 21,670 unit-hours. The computer program required 1.6 seconds of running time on a ADOSEA 486-33C computer to obtain this solution. This solution was found by searching 67 of the 206 possible nodes in the tree. This solution was confirmed by the complete enumeration computer code developed by Coutois et al.[2]. which checked all 120 possible tours.

Example 2

Given a set of 11 cities **N**, a matrix of times between these cities **T**, shown on the next page, a set of corresponding demands **D** and service rates **R**; find the tour through **N** which minimizes the average time for a unit of demand to be satisfied.

$$T = \begin{bmatrix} — & 25 & 50 & 54 & 35 & 29 & 19 & 298 & 303 & 12 & 307 \\ 18 & — & 20 & 36 & 15 & 54 & 124 & 132 & 148 & 137 & 156 \\ 25 & 18 & — & 33 & 187 & 44 & 25 & 12 & 20 & 35 & 29 \\ 26 & 24 & 37 & — & 22 & 22 & 112 & 107 & 9 & 125 & 115 \\ 27 & 39 & 47 & 121 & — & 50 & 118 & 14 & 105 & 135 & 126 \\ 33 & 33 & 48 & 40 & 50 & — & 126 & 18 & 113 & 109 & 116 \\ 54 & 150 & 23 & 132 & 165 & 145 & — & 74 & 90 & 69 & 108 \\ 65 & 116 & 122 & 207 & 127 & 131 & 46 & — & 69 & 77 & 84 \\ 97 & 14 & 123 & 115 & 99 & 108 & 36 & 43 & — & 48 & 36 \\ 38 & 201 & 195 & 116 & 21 & 115 & 3 & 33 & 30 & — & 54 \\ 47 & 119 & 14 & 56 & 131 & 116 & 24 & 28 & 6 & 42 & — \end{bmatrix}$$

$$D = \begin{bmatrix} 0 \\ 18 \\ 23 \\ 156 \\ 29 \\ 39 \\ 19 \\ 23 \\ 54 \\ 45 \\ 123 \end{bmatrix} \qquad R = \begin{bmatrix} 0 \\ .5 \\ .6 \\ .8 \\ .5 \\ .5 \\ .76 \\ .5 \\ .95 \\ 75 \\ 67 \end{bmatrix}$$

The optimal solution to this problem was found to be the tour 1-2-5-4-6-3-11-9-10-8-1 with a total area of 115,938

unit-hours. This solution was found on the ADOSEA 486-33C in 6 minutes and 13 seconds of computer. Because the program developed by Coutois et al.[2] can only handle problems of this size, no comparison could be made.

CHAPTER V

RESULTS, RECOMMENDATIONS, AND CONCLUSIONS

Results

Once the computer code validated the solution procedure, sample problems of different sizes were run to determine the computation time required as a function of problem size. The results shown in Figure 12 clearly show the an exponential relationship exists between problem size and computation time. This result was expected because almost all branch-and bound algorithms encounter this same result. However solving a problem by complete enumeration also exhibits an exponential relationship between problem size and computation time.

One interesting aspect of the results is that while the average computation time showed an exponential relationship, the variation between individual problems of the same size was very large. For example, one 10 node problem required just over 4 minutes while another problem of the same size required almost 8.5 minutes. Also one 20 node problem was solved in 1.5 minutes, however this problem was designed to achieve maximum fathoming. Therefore, the conclusion is the

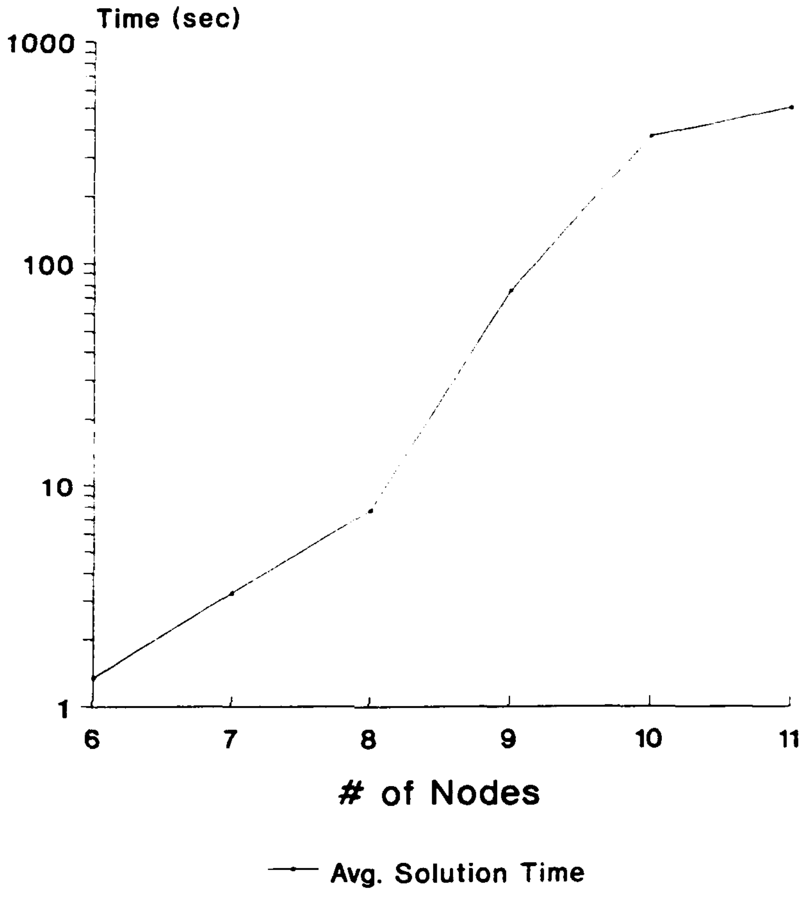


Figure 12 Computation Time vs. Problem Size

algorithm developed can be more efficient than complete enumeration. In general, the branch-and-bound search tree will contain approximately 71.8% more nodes than the number of feasible tours. However, the experience of this research found the branch-and-bound algorithm would generally reach optimality in less than the (n-1)! number of nodes searched.

The advantages of this solution approach are that less calculation is required in determining the lower bounds of subsets versus determining the total areas for all feasible tours. The primary disadvantage is that the computer code to implement the algorithm contains many loops and matrix re-initializations which take up substantial of computer time. These are a result of the algorithm searching for the maximum slope and demands remaining in the corresponding matrix and set. While this is a task that a human can perform quite easily, this researcher was unable to find a more efficient way for the computer to conduct these searches.

Recommendations

The branch-and-bound algorithm developed in this thesis has proven its ability to provide optimal solutions to problems of up to 20 cities. This algorithm should also provide solutions to very large problems, but the difficulty with exponentially increasing computation times may make this solution procedure impractical with problems of larger size. One consideration in accounting for the computation times shown in Figure 11 is the limited programming experience of the programmer. The first revision of the program which successfully implemented the solution procedure required an average of 1.5 minutes of computation time for problems with 6 cities. Additional effort in making the program more efficient reduced the average time for a 6 city problem to 1.5 seconds. The experience of the researcher also limited the program code to be written in GW-Basic which at this time is far from a state-of-the-art language. Future research might be directed towards coding the algorithm in a more efficient manner or with a program language better suited for this type of problem.

Another area for future research would be to develop a better procedure for determining the initial upper bound. The importance of finding a solution close to the optimal solution cannot be emphasized enough. Future research may be directed to finding a heuristic approach for larger problems. The results of a good heuristic could be incorporated as the procedure for determining the initial upper bound in the solution procedure developed in this thesis and thereby enhance the fathoming process and reduce computation time.

One further area of research which could be continued on this problem is to find a method which establishes a less conservative lower bound at any node. The results of this effort would also reduce the number of nodes required to be searched and would speed up the required computational time.

REFERENCES

[1]J.L. ARTHUR and J.O. FRENDEWEY (1988) Generating travelling-salesman problems with known optimal tours. *J. Opl. Res. Soc.* **39**, 153-159

[2]D.N. COUTOIS, K.P. GRANT, and A.J. KANE (1988) The search for a technique to identify the network tour which satisfies composite network demand at the optimal rate. An unpublished INEN 661 Project Report, Texas A&M University (Private Collection, A. Garcia-Diaz)

[3]D. PHILLIPS and A. GARCIA-DIAZ (1981) *Fundamentals of Network Analysis.* Prentice Hall, Englewood Cliffs, N.J.

[4]S.R. EVANS and J.P.NORBACK (1984) An heuristic method for solving time-sensitive routeing problems. *J. Opl. Res. Soc.* **35**, 407-414

[5]M. PADBERG and G. RINALDI (1991) A branch and cut algorithm for the resolution of large scale symmetric travelling salesman problems. *SIAM Review* **33**, 60-100

[6]E.L. Lawler, J.K. LENSTRA, A.H.G. RINNOYKAN and D.B. SHROYS (1985) *The Traveling Salesman Problem.* John Wiley & Sons, Chichester, England

[7]G. LAPORE, Y. NOBERT, and M. DESROCHERS (1984) Optimal routing under capacity and distance restrictions. *Opns. Res.* **33**, 1050-1073

[8]J.D. LITTLE, K.G. MURTY, D.W. SWEENEY, and C. KAREL (1963) An algorithm for the traveling salesman problem. *Opns. Res.* **11**, 972-989

[9]H. CROWDER and M.W. PADBERG (1980) Solving large-scale symmetric traveling salesman problems to optimality. *Management Sci.* **26**, 495-509

APPENDIX A

LISTING OF SOLUTION PROGRAM

```
10 REM
20 REM   THIS PROGRAM SOLVES THE TRAVELING SALESMAN PROBLEM
TO
30 REM   OPTIMIZE THE AVERAGE TIME TO CUSTOMER
SATISFACTION.
40 REM
50 REM
60 REM
70 REM   *** SETS THE DIMENSIONS OF MATRICES AND VECTORS
***
80 REM
90 DIM DIJ(20,20):DIM PATH(21):DIM INIT(20,20):DIM
SLOPE(20,20):DIM DISP(20*20,3):DIM SLOPE1(20,20):DIM
DEM(20):DIM TEMPDEM(20):DIM SLTEMP(20,20):DIM
PATHOPT(21):DIM PATH1(21):DIM TIME(20,20):DIM
SLTEMP2(20,20):NODES=0:HOLDER=0:HOLD2=0:HOLD3=0:HOLD4=0
100 KEY OFF:COLOR 7:NLINE=22:GOSUB 270
110 PGMN$="TSP FOR CUSTOMER DEMAND SATISFACTION"
120 LOCATE 1,INT(39-LEN(PGMN$)/2):PRINT
CHR$(181)+PGMN$+CHR$(198)
130 LOCATE 3,20:PRINT "MAKE A SELECTION"
140 LOCATE 5,20:PRINT "1) CREATE A NEW DATA SET"
150 LOCATE 6,20:PRINT "2) USE AN EXISTING DATA SET (FROM A
FILE)"
160 LOCATE 7,20:PRINT "3) GET A LISTING OF AVAILABLE FILES"
170 LOCATE 9,20:PRINT "4) RETURN TO GWBASIC"
180 LOCATE 10,20:PRINT "5) RETURN TO DOS"
190 LOCATE 12,20:INPUT "CHOOSE SELECTION:
",ANS$:ANS=VAL(ANS$)
200 IF ANS=1 THEN GOSUB 360:GOSUB 1020:GOSUB 1980:GOSUB
2830:GOSUB 3010:GOSUB 3650:GOSUB 4500:GOTO 100
210 IF ANS=2 THEN GOSUB 2430:GOSUB 1020:GOSUB 1980:GOSUB
2830:GOSUB 3010:GOSUB 3650:GOSUB 4500:GOTO 100
220 IF ANS=3 THEN GOSUB 1840:GOTO 100
230 IF ANS=4 THEN CLS:CLOSE:END
240 IF ANS=5 THEN CLS:SYSTEM
250 SOUND 500,1:SOUND 100,2:LOCATE 13,20:PRINT "INVALID
RESPONSE - PLEASE REENTER":GOTO 190
260 REM
270 REM *********** MAKES SCREEN FRAME *****************
280 REM
```

```
290 CLS:LOCATE 1,1:PRINT
CHR$(201)+STRING$(78,205)+CHR$(187)
300 FOR KLINE=1 TO NLINE
310 LOCATE KLINE+1:PRINT CHR$(186):LOCATE KLINE+1,80:PRINT
CHR$(186)
320 NEXT KLINE:LOCATE NLINE+1
330 PRINT CHR$(200)+STRING$(78,205)+CHR$(188)
340 RETURN
350 REM
360 REM ***************  CREATE NEW DATA SET
*****************
370 REM
380 CLS:NLINE=22:GOSUB 270
390 LOCATE 10,20:INPUT "HOW MANY NODES TO YOU HAVE? ",NODES
400 IF (NODES>20 OR NODES<3) THEN LOCATE 12,11:PRINT "MUST
BE AT LEAST 3 AND NO MORE THAN 20 NODES - PLEASE
REENTER":GOTO 390
410 FOR I=1 TO NODES
420 FOR J=1 TO NODES
430 DIJ(I,J)=999
440 NEXT J:NEXT I
450 CLS:NLINE=22:GOSUB 270
460 LOCATE 3,11:PRINT "TO ENTER DATA"
470 LOCATE 4,11:PRINT "TYPE THE NUMBER OF THE START NODE
AND HIT RETURN"
480 LOCATE 5,11:PRINT "TYPE THE NUMBER OF THE END NODE AND
HIT RETURN"
490 LOCATE 6,11:PRINT "TYPE THE CAPACITY ALONG THAT ARC AND
HIT RETURN"
500 LOCATE 8,11:PRINT "TO TERMINATE DATA ENTRY, TYPE A ZERO
FOR THE TWO NODE VALUES"
510 LOCATE 10,11:INPUT "PRESS RETURN TO CONTINUE",ANS$
520 CLS:NLINE=22:GOSUB 270:DISPLINE=6
530 LOCATE 2,32:PRINT "DATA INPUT"
540 LOCATE 4,26:PRINT "START   END      ARC":LOCATE
5,26:PRINT "NODE    NODE     LENGTH"
550 LOCATE DISPLINE,27:PRINT "   ":LOCATE DISPLINE,27:INPUT
"",ANS$:LOCATE 22,15:PRINT "
                           "
560 FOR I=1 TO LEN(ANS$)
570 ANS=ASC(MID$(ANS$,I,1))
580 IF (ANS<48 OR ANS>57) THEN LOCATE 22,15:PRINT
"CHARACTER IS NOT PERMITTED - PLEASE REENTER":GOTO 550
590 NEXT I
600 SNODE=VAL(ANS$)
610 IF (SNODE<0 OR SNODE>NODES) THEN LOCATE 22,15:PRINT
"NODE VALUE MUST BE BETWEEN 0 AND ";NODES;" - PLEASE
REENTER":GOTO 550
620 LOCATE DISPLINE,35:PRINT "   ":LOCATE DISPLINE,35:INPUT
```

```
"",ANS$:LOCATE 22,15:PRINT "
                            "
630 FOR I=1 TO LEN(ANS$)
640 ANS=ASC(MID$(ANS$,I,1))
650 IF (ANS<48 OR ANS>57) THEN LOCATE 22,15:PRINT
"CHARACTER IS NOT PERMITTED - PLEASE REENTER":GOTO 620
660 NEXT I
670 ENODE=VAL(ANS$)
680 IF (ENODE<0 OR ENODE>NODES) THEN LOCATE 22,15:PRINT
"NODE VALUE MUST BE BETWEEN 0 AND ";NODES;" - PLEASE
REENTER":GOTO 620
690 IF (SNODE=0 AND ENODE=0) THEN GOTO 810
700 IF SNODE=ENODE THEN LOCATE 22,15:PRINT "SELF LOOPS ARE
NOT ALLOWED - PLEASE REENTER END NODE":GOTO 620
710 LOCATE DISPLINE,45:PRINT "    ":LOCATE DISPLINE,45:INPUT
"",ANS$:LOCATE 22,15:PRINT "
                            "
720 FOR I=1 TO LEN(ANS$)
730 ANS=ASC(MID$(ANS$,I,1))
740 IF (ANS<48 OR ANS>57) THEN LOCATE 22,15:PRINT
"CHARACTER IS NOT PERMITTED - PLEASE REENTER":GOTO 710
750 NEXT I
760 ARCLEN=VAL(ANS$)
770 IF ARCLEN<0 THEN LOCATE 22,15:PRINT "NEGATIVE VALUES
ARE NOT PERMITTED - PLEASE REENTER":GOTO 710
780 DIJ(SNODE,ENODE)=ARCLEN
790 IF DISPLINE<15 THEN DISPLINE=DISPLINE+1:GOTO 550
800 GOTO 520
810 FOR I=1 TO NODES
820 DIJ(I,0)=0
830 FOR J=1 TO NODES
840 IF DIJ(I,J)>=999 GOTO 860
850 DIJ(I,0)=DIJ(I,0)+DIJ(I,J)
860 NEXT J
870 NEXT I
880 LOCATE 22,11:INPUT "PRESS RETURN TO CONTINUE",ANS$
890 CLS:NLINE=22:GOSUB 270:DISPLINE=6
900 LOCATE 2,32:PRINT "DEMAND DATA"
910 LOCATE 4,26:PRINT "NODE     DEMAND"
920 FOR I=1 TO NODES
930 LOCATE DISPLINE,27:PRINT I
940 LOCATE DISPLINE,36:PRINT "    ":LOCATE DISPLINE,36:INPUT
"",ANS$:LOCATE 22,15:PRINT "
                             "
950 ANS = VAL(ANS$)
960 IF ANS<0 THEN LOCATE 22,15:PRINT "NEGATIVE VALUES ARE
NOT PERMITTED - PLEASE REENTER":GOTO 930
970 DEM(I) = ANS
980 DISPLINE = DISPLINE+1
```

```
990 NEXT I
1000 RETURN
1010 REM
1020 REM **************** EDIT DATA *****************
1030 REM
1040 CLS:NLINE=22:GOSUB 270
1050 LOCATE 3,11:INPUT "DO YOU WANT TO EDIT THE INPUT DATA
(Y/N)? ",YN$
1060 YN=ASC(MID$(YN$,1,1))
1070 IF (YN=78 OR YN=110) THEN GOTO 1590
1080 IF (YN=89 OR YN=121) THEN GOTO 1100
1090 LOCATE 22,15:PRINT "INVALID RESPONSE - PLEASE
REENTER":GOTO 1050
1100 ARCNUM=1
1110 FOR I=1 TO NODES
1120  FOR J=1 TO NODES
1130 IF DIJ(I,J)>=999 GOTO 1150
1140
DISP(ARCNUM,1)=I:DISP(ARCNUM,2)=J:DISP(ARCNUM,3)=DIJ(I,J):A
RCNUM=ARCNUM+1
1150 NEXT J
1160 NEXT I
1170 ARCKOUNT=1
1180 CLS:NLINE=18:GOSUB 270
1190 LOCATE 2,32:PRINT "EDIT DATA"
1200 LOCATE 4,19:PRINT "ARC    START   END      ARC":LOCATE
5,19:PRINT "NUMBER NODE    NODE      LENGTH"
1210 FOR DISPLINE=6 TO 15
1220 IF ARCKOUNT=>ARCNUM GOTO 1250
1230 LOCATE DISPLINE,20:PRINT ARCKOUNT:LOCATE
DISPLINE,27:PRINT DISP(ARCKOUNT,1):LOCATE DISPLINE,35:PRINT
DISP(ARCKOUNT,2):LOCATE DISPLINE,45:PRINT
DISP(ARCKOUNT,3):ARCKOUNT=ARCKOUNT+1
1240 NEXT DISPLINE
1250 LOCATE 21,15:INPUT "DO YOU WANT TO CHANGE ANY OF THE
ABOVE INFORMATION? ",YN$
1260 YN=ASC(MID$(YN$,1,1))
1270 IF (YN=78 OR YN=110) THEN GOTO 1580
1280 IF (YN=89 OR YN=121) THEN GOTO 1300
1290 LOCATE 22,15:PRINT "INVALID RESPONSE - PLEASE
REENTER":GOTO 1250
1300 LOCATE 21,15:INPUT "DO YOU WANT TO ADD (A), DELETE (D)
OR CHANGE (C) AN ARC? ",ANS$
1310 ANS=ASC(MID$(ANS$,1,1))
1320 IF (ANS=65 OR ANS=97) THEN GOTO 1440
1330 IF (ANS=68 OR ANS=100) THEN GOTO 1410
1340 IF (ANS<>67 AND ANS<>99) THEN LOCATE 22,15:PRINT
"INVALID RESPONSE - PLEASE REENTER":GOTO 1300
1350 LOCATE 21,15:PRINT "
```

```
                              ":LOCATE 21,15:INPUT "WHICH ARC TO
CHANGE? ",ANS$
1360 ANS=VAL(ANS$)
1370 LOCATE 21,15:PRINT "START NODE = ",DISP(ANS,1),"  END
NODE = ",DISP(ANS,2)
1380 LOCATE 22,15:PRINT "CURRENT LENGTH = ",DISP(ANS,3)
1390 LOCATE 23,15:INPUT "WHAT IS THE NEW ARC LENGTH?
",ANS1$
1400 ANS1=VAL(ANS1$):DIJ(DISP(ANS,1),DISP(ANS,2))=ANS1:GOTO
1480
1410 LOCATE 21,15:PRINT "
                             ":LOCATE 21,15:INPUT "WHICH ARC TO
DELETE? ",ANS$
1420 ANS=VAL(ANS$)
1430 DIJ(DISP(ANS,1),DISP(ANS,2))=999:GOTO 1480
1440 LOCATE 21,15:PRINT "
                             ":LOCATE 21,15:INPUT "START NODE =
",SNODE
1450 LOCATE 22,15:INPUT "END NODE = ",ENODE
1460 LOCATE 23,15:INPUT "ARC LENGTH = ",ARCLEN
1470 DIJ(SNODE,ENODE)=ARCLEN
1480 ARCNUM=1
1490 FOR I=1 TO NODES
1500 DIJ(I,0)=0
1510 FOR J=1 TO NODES
1520 IF INIT(I,J)=999 GOTO 1550
1530 DIJ(I,0)=DIJ(I,0)+DIJ(I,J)
1540
DISP(ARCNUM,1)=I:DISP(ARCNUM,2)=J:DISP(ARCNUM,3)=DIJ(I,J):A
RCNUM=ARCNUM+1
1550 NEXT J
1560 NEXT I
1570 ARCKOUNT=ARCKOUNT-10:GOTO 1180
1580 IF ARCKOUNT<ARCNUM THEN GOTO 1180
1590 CLS:NLINE=22:GOSUB 270
1600 LOCATE 3,11:INPUT "DO YOU WANT TO EDIT THE DEMAND DATA
(Y/N)? ",YN$
1610 YN=ASC(MID$(YN$,1,1))
1620 IF (YN=78 OR YN=110) THEN RETURN
1630 IF (YN=89 OR YN=121) THEN GOTO 1650
1640 LOCATE 22,15:PRINT "INVALID RESPONSE - PLEASE
REENTER":GOTO 1590
1650 CLS:NLINE=22:GOSUB 270
1660 LOCATE 4,19:PRINT "NODE       DEMAND           "
1670 FOR DISPLINE = 1 TO NODES
1680 LOCATE DISPLINE+4,20:PRINT DISPLINE:LOCATE
DISPLINE+4,32:PRINT DEM(DISPLINE)
1690 NEXT DISPLINE
1700 LOCATE 21,15:INPUT "DO YOU WANT TO CHANGE ANY OF THE
```

```
ABOVE INFORMATION? ",YN$
1710 YN=ASC(MID$(YN$,1,1))
1720 IF (YN=78 OR YN=110) THEN GOTO 1820
1730 IF (YN=89 OR YN=121) THEN GOTO 1750
1740 LOCATE 22,15:PRINT "INVALID RESPONSE - PLEASE
REENTER:GOTO 1700
1750 LOCATE 21,15:PRINT "
                       ":LOCATE 21,15:INPUT "WHICH NODE TO
CHANGE? ,ANS$
1760 ANS=VAL (ANS$)
1770 LOCATE 21,15:PRINT "
 ":LOCATE 21,15:PRINT "NODE = ",ANS
1780 LOCATE 22,15:PRINT "CURRENT DEMAND = "DEM(ANS)
1790 LOCATE 23,15:INPUT "WHAT IS THE NEW DEMAND? "ANS1$
1800 ANS1=VAL (ANS1$):DEM(ANS)=ANS1
1810 GOTO 1650
1820 RETURN
1830 REM
1840 REM ************** DIRECTORY OF EXISTING FILES
**************
1850 REM
1860 PGMNAME$="TSP FOR CDS"
1870 FLE$="*.CDS"
1880 LOCATE 13,5:INPUT "ENTER COMPLETE PATH:  ",PATH$:CLS
1890 IF LEN(PATH$)=2 AND MID$(PATH$,2,1)=":" THEN 1920
1900 IF LEN(PATH$)>0 AND RIGHT$(PATH$,1)<>"\" THEN
PATH$=PATH$+"\"
1910 IF LEN(PATH$)>0 AND MID$(PATH$,2,1)<>":" THEN
PATH$="\"+PATH$
1920 LOCATE 19,11:PRINT PGMNAME$;" FILES ON PATH ";PATH$
1930 LOCATE 20,2:FILES PATH$+FLE$
1940 LOCATE 24,1:INPUT "PRESS ANY KEY TO CONTINUE",ANS$
1950 CLS
1960 RETURN
1970 REM
1980 REM ************* SAVE DATA ******************
1990 REM
2000 CLS:NLINE=22:GOSUB 270
2010 LOCATE 3,11:INPUT "DO YOU WANT TO SAVE THE INPUT DATA
(Y/N)? ",YN$
2020 YN=ASC(MID$(YN$,1,1))
2030 IF (YN=78 OR YN=110) THEN RETURN
2040 IF (YN=89 OR YN=121) THEN GOTO 2060
2050 LOCATE 22,15:PRINT "INVALID RESPONSE - PLEASE
REENTER":GOTO 2010
2060 IF NODES<=0 THEN LOCATE 22,15:INPUT "NO DATA AVAILABLE
- PRESS RETURN TO CONTINUE",ANS$:RETURN
2070 CLS:NLINE=22:GOSUB 270
2080 FLE$=".CDS"
```

```
2090 LOCATE 13,5:INPUT "ENTER COMPLETE PATH:  ",PATH$:CLS
2100 IF LEN(PATH$)=2 AND MID$(PATH$,2,1)=":" THEN 2130
2110 IF LEN(PATH$)>0 AND RIGHT$(PATH$,1)<>"\" THEN
PATH$=PATH$+"\"
2120 IF LEN(PATH$)>0 AND MID$(PATH$,2,1)<>":" THEN
PATH$="\"+PATH$
2130 LOCATE 14,5:INPUT "ENTER A FILE NAME (WITHOUT
EXTENSION) - ",FILE$
2140 X=INSTR(FILE$,".")
2150 LOCATE 22,15:IF LEN(FILE$)=0 OR X=1 THEN PRINT
"INVALID FILE NAME - PLEASE REENTER":GOTO 2130
2160 IF X>0 THEN FILE$=LEFT$(FILES,X-1)
2170 FILENAME$=PATH$+FILE$+FLE$
2180 PRINT FILENAME$:INPUT "HIT RETURN",ANS$
2190 ON ERROR GOTO 2380:OPEN "I",#1,FILENAME$
2200 CLOSE #1:LOCATE 16,15:INPUT "FILE ALREADY EXISTS -
OVERWRITE? (Y/N) ",YN$
2210 YN=ASC(MID$(YN$,1,1))
2220 IF (YN=78 OR YN=110) THEN LOCATE 16,15:PRINT "
                                              ":GOTO 2130
2230 IF (YN=89 OR YN=121) THEN GOTO 2250
2240 IF (YN<>89 OR YN<>121) THEN LOCATE 22,15:PRINT
"INVALID RESPONSE - PLEASE REENTER":GOTO 2200
2250 CLOSE #1:OPEN "O",#1,FILENAME$
2260 WRITE#1,NODES
2270 FOR I=1 TO NODES
2280 WRITE#1,DEM(I)
2290 NEXT I
2300 FOR I=1 TO NODES
2310 FOR J=1 TO NODES
2320 IF DIJ(I,J)=999 GOTO 2340
2330 WRITE#1,I,J,DIJ(I,J)
2340 NEXT J
2350 NEXT I
2360 WRITE#1,"0","0","0"
2370 CLOSE #1
2380 REM *** ERROR TRAPPING ***
2390 IF ERR=53 AND ERL=2190 THEN RESUME 2250
2400 ON ERROR GOTO 0
2410 RETURN
2420 REM
2430 REM *************** RETRIEVE DATA ****************
2440 CLS:NLINE=22:GOSUB 270
2450 FLE$=".CDS"
2460 LOCATE 13,5:INPUT "ENTER COMPLETE PATH:  ",PATH$:CLS
2470 IF LEN(PATH$)=2 AND MID$(PATH$,2,1)=":" THEN 2500
2480 IF LEN(PATH$)>0 AND RIGHT$(PATH$,1)<>"\" THEN
PATH$=PATH$+"\"
2490 IF LEN(PATH$)>0 AND MID$(PATH$,2,1)<>":" THEN
```

```
PATH$="\"+PATH$
2500 LOCATE 14,5:INPUT "ENTER A FILE NAME (WITHOUT
EXTENSION) - ",FILE$
2510 X=INSTR(FILE$,".")
2520 LOCATE 22,15:IF LEN(FILE$)=0 OR X=1 THEN PRINT
"INVALID FILE NAME - PLEASE REENTER":GOTO 2500
2530 IF X>0 THEN FILE$=LEFT$(FILES,X-1)
2540 FILENAME$=PATH$+FILE$+FLE$
2550 ON ERROR GOTO 2780:OPEN FILENAME$ FOR INPUT AS #1
2560 INPUT #1,NODES
2570 FOR I=1 TO NODES
2580 INPUT #1,DEM(I)
2590 NEXT I
2600 FOR I=1 TO NODES
2610 FOR J=1 TO NODES
2620 DIJ(I,J)=999
2630 NEXT J
2640 NEXT I
2650 WHILE NOT EOF(1)
2660 INPUT #1,SNODE,ENODE,ARCLEN
2670 DIJ(SNODE,ENODE)=ARCLEN
2680 WEND
2690 FOR I=1 TO NODES
2700 DIJ(I,0)=0
2710 FOR J=1 TO NODES
2720 IF DIJ(I,J)=999 GOTO 2740
2730 DIJ(I,0)=DIJ(I,0)+DIJ(I,J)
2740 NEXT J
2750 NEXT I
2760 CLOSE #1:GOTO 2810
2770 CLOSE #1:LOCATE 16,15:PRINT "FILE DOES NOT EXIST -
PLEASE REENTER ":GOTO 2500
2780 REM *** ERROR TRAPPING ***
2790 IF ERR=53 AND ERL=2550 THEN RESUME 2770
2800 ON ERROR GOTO 0
2810 RETURN
2820 REM
2830 REM ********** INITIALIZE CONDITIONS FOR ALGORITHM
*************
2840 REM
2850 INIT(1,1)=9999999!
2860 SLOPE(1,1)=0
2870 FOR I=2 TO NODES
2880 INIT(I,1)=0
2890 SLOPE(I,1)=0
2900 NEXT I
2910 FOR I=1 TO NODES
2920 FOR J=2 TO NODES
2930 INIT(I,J)=DIJ(I,J)
```

```
2940 IF I=J THEN DIJ(I,J)=9999999!
2950 SLOPE(I,J)=DEM(J)/INIT(I,J)
2960 IF I=J THEN SLOPE(I,J)=0
2970 NEXT J
2980 NEXT I
2990 RETURN
3000 REM
3010 REM ************** PERFORM THE TWO-STEP GREEDY
ALGORITHM *********
3020 REM ***************** TO FIND THE INITIAL UPPER BOUND
************
3030 REM
3040 FOR I=1 TO NODES
3050 FOR J=1 TO NODES
3060 SLOPE1(I,J)=SLOPE(I,J)
3070 NEXT J
3080 NEXT I
3090 FOR I=1 TO NODES+1
3100 PATH(I)=0
3110 NEXT I
3120 PATH(1) = 1:PATH(NODES+1)=1
3130 GREED1=0
3140 GREED2=0
3150 COUNT=1
3160 I=1
3170 FOR J=2 TO NODES
3180 IF J=I THEN GOTO 3280
3190 FOR K=2 TO NODES
3200 IF K=J THEN GOTO 3230
3210 IF SLOPE1(J,K) >= GREED1 THEN GREED1 = SLOPE1(J,K)
ELSE GOTO 3230
3220 HOLD1 =K
3230 NEXT K
3240 IF SLOPE1(I,J)+GREED1 >= GREED2 THEN
GREED2=SLOPE1(I,J)+GREED1 ELSE GOTO 3280
3250 HOLD2=J
3260 HOLD3=HOLD1
3270 GREED1=0
3280 NEXT J
3290 PATH(COUNT+1)=HOLD2
3300 PATH(COUNT+2)=HOLD3
3310 COUNT=COUNT+2
3320 GREED2=0
3330 FOR I=1 TO NODES
3340 SLOPE1(1,I)=0
3350 SLOPE1(I,HOLD2)=0
3360 SLOPE1(HOLD2,I)=0
3370 SLOPE1(I,HOLD3)=0
3380 NEXT I
```

```
3390 IF PATH(NODES-1)=0 THEN I=HOLD3:GOTO 3170 ELSE GOTO
3400
3400 FOR I=2 TO NODES
3410 FOR J=2 TO NODES
3420 IF I=PATH(J) GOTO 3450
3430 NEXT J
3440 PATH(NODES) = I
3450 NEXT I
3460 REM FOR I=1 TO NODES+1
3470 REM LOCATE I+3,11:PRINT PATH(I)
3480 REM NEXT I
3490 REM
3500 REM ********** CALCULATE INITIAL UPPER BOUND USING
*************
3510 REM ********** PATH GENERATED FROM TWO-STEP GREEDY
*************
3520 REM
3530 UB=0
3540 UNSAAT=0
3550 FOR I=1 TO NODES
3560 UNSAT = UNSAT + DEM(I)
3570 NEXT I
3580 FOR I=1 TO NODES
3590 UB= UB + INIT(PATH(I),PATH(I+1))*UNSAT
3600 UNSAT=UNSAT - DEM(PATH(I+1))
3610 NEXT I
3620 REM LOCATE 18,11:PRINT "UPPERBOUND = ",UB
3630 REM LOCATE 20,11:INPUT "PRESS RETURN TO CONTINUE",ANS$
3640 RETURN
3650 REM
3660 REM ************ TREE SEARCH SUBROUTINE
***************
3670 REM
3680 REM
3690 LBOPT =UB
3700 FOR I=1 TO NODES+1
3710 PATHOPT(I)=PATH(I)
3720 NEXT I
3730 NEWNODE=1
3740 COUNT=1
3750 FOR I=1 TO NODES
3760 PATH1(I)=0
3770 NEXT I
3780 PATH1(1)=1:PATH1(NODES+1)=1
3790 FOR I=1 TO NODES
3800 FOR J=1 TO NODES
3810 SLTEMP(I,J)=SLOPE(I,J)
3820 NEXT J
3830 NEXT I
```

```
3840 MAXSLOPE =0
3850 UNSAT=0
3860 FOR I=1 TO NODES
3870 UNSAT=UNSAT+DEM(I)
3880 TEMPDEM(I) = DEM(I)
3890 FOR J=1 TO NODES
3900 TIME(I,J)=INIT(I,J)
3910 SLTEMP2(I,J)=SLOPE(I,J)
3920 NEXT J
3930 NEXT I
3940 FOR I=COUNT+1 TO NODES
3950 PATH1(I)=0
3960 NEXT I
3970 FOR I=2 TO NODES
3980 FOR J=2 TO NODES
3990 IF PATH1(J)=I GOTO 4040
4000 NEXT J
4010 IF SLTEMP(NEWNODE,I)>MAXSLOPE THEN
MAXSLOPE=SLTEMP(NEWNODE,I)ELSE GOTO 4040
4020 IF COUNT<NODES THEN PATH1(COUNT+1)=I
4030 HOLDER=0
4040 NEXT I
4050 IF MAXSLOPE=0 AND COUNT=1 THEN RETURN
4060 IF MAXSLOPE=0 AND HOLDER>0 THEN
COUNT=COUNT-HOLDER:NEWNODE=PATH1(COUNT):GOSUB
4630:SLTEMP(NEWNODE,PATH1(COUNT+1))=0:PATH1(COUNT+1)=0:GOTO
3840
4070 IF MAXSLOPE=0 THEN
COUNT=COUNT-1:NEWNODE=PATH1(COUNT):IF COUNT=1 THEN GOSUB
4610:SLTEMP(NEWNODE,PATH1(COUNT+1))=0:PATH1(COUNT+1)=0:HOLD
ER=HOLDER+1:GOTO 3840 ELSE
SLTEMP(NEWNODE,PATH1(COUNT+1))=0:PATH1(COUNT+1)=0:HOLDER=HO
LDER+1:GOTO 3840
4080 LB=0
4090 FOR J=1 TO COUNT
4100 IF PATH1(J+1)=0 THEN GOTO 420
4110 TEMPDEM(PATH1(J))=0
4120 LB = LB +TIME(PATH1(J),PATH1(J+1))*UNSAT
4130 UNSAT=UNSAT-TEMPDEM(PATH1(J+1))
4140 FOR K=1 TO NODES
4150 SLTEMP2(PATH1(J),K)=0
4160 SLTEMP2(K,PATH1(J+1))=0
4170 NEXT K
4180 TEMPDEM(PATH1(J+1))=0
4190 NEXT J
4200 FOR L=1 TO NODES
4210 IF PATH1(L) > 0 THEN TIME(PATH1(COUNT+1),PATH1(L)) =
9999999!
4220 NEXT L
```

```
4230 MINTIME = 9999999!
4240 FOR M=1 TO NODES
4250 IF TIME(PATH1(COUNT+1),M)<MINTIME THEN
MINTIME=TIME(PATH1(COUNT+1),M)
4260 NEXT M
4270 LB=LB+MINTIME*UNSAT
4280 FOR K=COUNT+2 TO NODES-1
4290 GOSUB 4730:GOSUB 4840
4300 REM LOCATE 10,20:PRINT UNSAT-MAXDEM1
4310 REM LOCATE LOCATE 11,20:PRINT MAXSLOPE2
4320 IF MAXDEM2>0 THEN
LB=LB+(UNSAT-MAXDEM1)*MAXDEM2/MAXSLOPE2 ELSE LB =
LB+MAXDEM1*MAXDEM1/MAXSLOPE2
4330 UNSAT =UNSAT-MAXDEM1
4340 NEXT K
4350 REM LOCATE 9,13:PRINT LB
4360 REM LOCATE 10,13:PRINT NEWNODE
4370 REM LOCATE 11,13:PRINT PATH1(COUNT+1)
4380 REM LOCATE 20,11:PRINT "PRESS RETURN TO CONTINUE",ANS$
4390 IF LB>= LBOPT THEN
SLTEMP(NEWNODE,PATH1(COUNT+1))=0:PATH1(COUNT+1)=0 ELSE GOTO
4410
4400 GOTO 3840
4410 NEWNODE=PATH1(COUNT+1)
4420 COUNT=COUNT+1
4430 IF PATH1(NODES)>0 THEN LBOPT=LB ELSE GOTO 3840
4440 FOR I=1 TO NODES
4450 PATHOPT(I) =PATH1(I)
4460 NEXT I
4470 GOTO 3840
4480 REM ******** END SUBROUTINE ******************
4490 REM
4500 REM******** OUTPUT SUBROUTINE *****************
4510 REM
4520 SHOW=7
4530 CLS:GOSUB 270
4540 LOCATE SHOW,11:PRINT "OPTIMAL AREA = ",LBOPT
4550 FOR I=1 TO NODES+1
4560 LOCATE SHOW+I,11:PRINT PATHOPT(I)
4570 NEXT I
4580 LOCATE 20,11:INPUT "PRESS RETURN TO CONTINUE",ANS$
4590 RETURN
4600 REM
4610 REM ********* SUBROUTINE TO REINITIALIZE SLOPE VALUES
*****
4620 REM
4630 FOR I=1 TO NODES
4640 FOR J=1 TO NODES
4650 SLTEMP(I,J)=SLOPE(I,J)
```

```
4660 NEXT J
4670 NEXT I
4680 FOR I=1 TO NODES
4690 IF SLTEMP(NEWNODE,I) > SLTEMP(NEWNODE,PATH1(COUNT+1))
THEN SLTEMP(NEWNODE,I)=0
4700 NEXT I
4710 RETURN
4720 REM
4730 REM ********* SUBROUTINE TO FIND THE MAX SLOPE
**********
4740 REM
4750 MAXSLOPE2=0:HOLD3=0:HOLD4=0
4760 FOR I=1 TO NODES
4770 FOR J=1 TO NODES
4780 IF SLTEMP2(I,J)>MAXSLOPE2 THEN
MAXSLOPE2=SLTEMP(I,J):HOLD3=I:HOLD4 = J
4790 NEXT J
4800 NEXT I
4810 SLTEMP2(HOLD3,HOLD4) =0
4820 RETURN
4830 REM
4840 REM *********** SUBROUTINE TO FIND THE MAXIMUM
**********
4850 REM *************** UNSATISFIED DEMANDS
*****************
4860 REM
4870 MAXDEM1=0:MAXDEM2=0
4880 FOR I=1 TO NODES
4890 IF TEMPDEM(I)>=MAXDEM1 THEN
MAXDEM2=MAXDEM1:MAXDEM1=TEMPDEM(I):HOLD2=I ELSE IF
TEMPDEM(I)>MAXDEM2 THEN MAXDEM2=TEMPDEM(I)
4900 NEXT I
4910 TEMPDEM(HOLD2)=0
4920 REM LOCATE 10,17:PRINT MAXDEM1
4930 REM LOCATE 11,17:PRINT MAXDEM2
4940 REM LOCATE 20,11:INPUT "PRESS RETURN TO CONTINUE",ANS$
4950 RETURN
```

APPENDIX B

LIST OF USER INSTRUCTIONS

Starting Up the Program

To start up the computer program place the disk containing the program into Drive A and at the DOS prompt type "A:DEMSAT" and press RETURN. The main menu will appear which allows you to choose between creating a new set of data, loading an existing set of data, and getting a directory of existing data files.

Creating a New Set of Data

If you choose to create a new set of data the screen will show three column headings: Source Node, End Node, and Arc Length. To enter the data, type the source node of an arc and press RETURN, then type the terminal node and press RETURN, next type the distance between the nodes (cities) and press RETURN. The program will automatically move the cursor to the required position after each time RETURN is pressed. Continue to insert all your data in the following manner. When you have entered all your data, type "0" and press RETURN for the first two columns and the next menu will appear.

The next menu asks you to input the demand data, the menu lists each node number consecutively, and you are required to type the demand for that node and press RETURN. Following the demand data screen is the service rate data screen. Service rates are entered in the same manner as demand data. Remember that the demand and service rate at city 1 is assumed to be 0, therefore you should enter it as such.

After you have entered all the data, the program will ask if you would like to edit ant of the data. If you choose to do so the program will guide you through the editing by asking appropriate questions and prompting you for answers.

The program will next ask if you would like to save the data set. If you choose to do so, The program will prompt you for a path to save the data to (i.e. "A:") and then ask for the filename (i.e. "DATA1"). All files will be saved with ".CDS" attached to the filename (i.e. "DATA1.CDS").

The program will then run the data a present the solution on the screen. When you are done viewing the solution, press RETURN and the program will return you to the DOS prompt.

Loading Existing Data

If at the main menu you choose to run the program with existing data, the program will first prompt you for the path of the data (i.e. "A:") and then prompt you for the filename. The program will load the data from the file specified and then ask you if you wish to edit the data. From this point, the program runs as if you have just created the data, so the procedures are the same as above.

Directory of Existing Files

If you choose to view a directory of existing data files from the main menu, the program will prompt you for the path for the data files (i.e. "A:"). At this point, the program will search that path and list all files which end with ".CDS". After viewing the list press RETURN and the program will return you to the main menu.

VITA

James Patrick Ryan was born in New York City on September 21, 1963. His parents are James Joseph Ryan and Elizabeth Mary (White) Ryan. In 1981, James received a congressional appointment to the United States Air Force Academy and graduated with a B.S. on May 28, 1986. Upon graduating, James was commissioned a 2nd Lieutenant in the U.S. Air Force and was assigned for flying training at Columbus Air Force Base (AFB), Mississippi. It was there that he met Christine Marie Cabrera, and they were married on September 17, 1988. His next assignment was to Wright-Patterson AFB where he served as a manufacturing engineer in the Propulsion Systems Program Office. From January 1987 - May 1988, James was a program manager for the Air Force Industrial Modernization Incentives Program for the modernization of manufacturing facilities of two major Air Force jet engine manufacturers. From May 1988 - July 1990, James was responsible for the transition of a major fighter jet engine from the development phase to full scale production. In August 1990, was assigned to Texas A&M University to obtain a Masters Degree in Industrial Engineering. James is currently a Captain in the U.S. Air Force and a member of the Alpha Pi Mu Honor Society. His permanent address is 2811 S.E. 18 Ave Cape Coral FL 33904.